conference

proceedings

The Ninth Systems Administration Conference

LISA '95

Monterey, California
September 17–22, 1995

Sponsored by USENIX, the UNIX and Advanced Computing Systems Technical and Professional Association and SAGE, the System Administrators Guild

USENIX® SAGE® THE SYSTEM ADMINISTRATORS GUILD

USENIX The Ninth Systems Administration Conference LISA '95 Monterey, California September 1995

For additional copies of these proceedings contact

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710 USA
Telephone: 510-528-8649

The price is $30 for members and $38 for nonmembers.

Outside the U.S.A. and Canada, please add
$11 per copy for postage (via air printed matter).

Past USENIX Large Installation Systems Administration Workshop and Conference Proceedings (price: member/nonmember)

Large Installation Systems Admin. I Workshop	1987	Phildelphia, PA	$4/$4
Large Installation Systems Admin. II Workshop	1988	Monterey, CA	$8/$8
Large Installation Systems Admin. III Workshop	1989	Austin, TX	$13/$13
Large Installation Systems Admin. IV Conference	1990	Colorado Spgs, CO	$15/$18
Large Installation Systems Admin. V Conference	1991	San Diego, CA	$20/$23
Large Installation Systems Admin. VI Conference	1992	Long Beach, CA	$23/$30
Large Installation Systems Admin. VII Conference	1993	Monterrey, CA	$25/$33
Large Installation Systems Admin. VIII Conference	1994	San Diego, CA	$22/$29

Outside the U.S.A. and Canada, please add $10
per copy for postage (via air printed matter).

ISBN 1-880446-73-1

Printed in the United States of America on 50% recycled paper, 10-15% post consumer waste.

USENIX Association

Proceedings of the Ninth Systems Administration Conference (LISA IX)

September 17-22, 1995
Monterey, CA, USA

TABLE OF CONTENTS

Internet Services

Thursday (11:00-12:30) **Chair: Tina Darmohray**

Management

Thursday (2:00-3:30) **Chair: Kim Trudel**

User Environment

Thursday (4:00-5:30) **Chair: Bryan McDonald**

File Management

Friday (9:00-10:30) **Chair: Paul Anderson**

Network Hardware

Friday (11:00-12:30) — Chair: Paul Evans

ACKNOWLEDGMENTS

PROGRAM CO-CHAIRS
Tina M. Darmohray, *Information Warehouse!, Inc.*
Paul Evans, *Synopsys, Inc.*

PROGRAM COMMITTEE
Paul Anderson, *University of Edinburgh*
Rob Kolstad, *Berkeley Software Design, Inc.*
Bryan McDonald, *SRI International*
Marcus J. Ranum, *Information Warehouse!, Inc.*
John Schimmel, *Silicon Graphics, Inc.*
Kim Trudel, *Massachusetts Institute of Technology*

INVITED TALKS COORDINATOR
Laura de Leon, *Hewlett-Packard*
Peg Schafer, *Harvard University*

WORK-IN-PROGRESS COORDINATOR
Peg Schafer, *Harvard University*

GURU IS IN COORDINATOR
Lee Damon, *QUALCOMM, Inc.*

TUTORIAL COORDINATOR
Daniel V. Klein, *USENIX Association*

TERMINAL ROOM COORDINATOR
Henry Spencer, *SP Systems*

PROCEEDINGS PRODUCTION
Rob Kolstad, *Berkeley Software Design, Inc.*
Carolyn S. Carr, *USENIX Association*
Data Reproduction

USENIX MEETING PLANNER
Judith F. Desharnais, *USENIX Association*

USENIX EXECUTIVE DIRECTOR
Ellie Young, *USENIX Association*

USENIX SUPPORT STAFF
Colleen Biddle, *USENIX Association*
Eileen Curtis, *USENIX Association*
Diane DeMartini, *USENIX Association*
Toni Veglia, *USENIX Association*

USENIX Marketing Director
Zanna Knight, *USENIX Association*

USENIX EXHIBITS COORDINATOR
Peter Mui, *USENIX Assocation*

PREFACE

Welcome to LISA 95! We have an eventful week planned in Monterey for folks interested in system administration. Along with a high-quality refereed paper track, we have varied invited talks to highlight emerging technologies of special interest, as well as focused sessions for various levels of expertise. We hope you find what you're looking for in the offerings at this conference, and maybe some welcome surprises! Again, welcome!

We would like to thank our program committee members, who did an outstanding job of recruiting authors for the refereed paper track and then read through the resulting submissions in a little less than two weeks. We'd also like to thank Peg Schafer and Helen Harrison, who though not members of the program committee, served as paper reviewers and participated in the committee meeting.

The program committee would like to thank all those who submitted abstracts for the refereed paper track. It is a sign of the maturing both of this conference, and the profession for which it is the leading forum, that the program committee received such a large number of very high quality submissions.

We'd also like to thank Laura de Leon and Peg Schafer, who put together the Invited Talk track and the Works In Progress session as well as Lee Damon who coordinated both the Birds of a Feather sessions and the Guru Is In track.

We'd like to thank the USENIX office staff, both in Berkeley and at the Conference office in Lake Forest. Their professionalism and experience were a priceless asset to us in putting our piece of the conference together.

Finally, we'd like to encourage all system administrators to consider submitting their work for next year's conference – this is your conference, and its quality depends directly on the quality and quantity of submissions that the program committee receives from you!

Enjoy the conference!

Tina Darmohray
Paul Evans

AUTHOR INDEX

lbnamed: A Load Balancing Name Server in Perl

Roland J. Schemers, III – SunSoft, Inc.

ABSTRACT

Given a cluster of workstations, users have always wanted a way to login to the least-loaded workstation. This paper discusses an attempt to solve that problem using a load balancing name server. This name server also has the ability to serve other dynamic information as well, such as /etc/passwd information (a la Hesiod [2]). The prototype was written in Perl 4 [1], and recently converted to Perl 5. This paper describes the Perl 4 version first and then describes some of the interesting features in the Perl 5 version. This paper assumes the reader has a basic understanding of Perl, DNS, and BIND [3].

The Problem

When I joined the Distributed Computing Operations (DCO) group at Stanford, some of the machines in the public UNIX workstation clusters were overloaded but others were idle. People logging in remotely picked the same machine all the time, or they always picked the same architecture. For example, they would always login to a system running SunOS. If their favorite host was down, they would call the operations staff and complain that the whole system was down.

Users constantly asked for a way to login to the "best" workstation. The best answer consultants could give them was to login to some workstation, then run a program called "sweetload" which would return a sorted list of the loads on all the workstations. The user would then have to pick one workstation from the list, and login to it, possibly logging out of the workstation they ran sweetload on. This was not an ideal solution.

It was at that point I decided to start working on a "real" load balancing name server. I was interested in creating a DNS name server that could receive a request and dynamically create the response.

Stanford has a wide range of machines in the public workstation cluster(s). Figure 1 shows the diversity of machines. A user can login to any one of the workstations and see essentially the same environment: same home directory, mail, etc. This type of environment is well-suited for a load balancing name server.

19 Sparc 2s
37 Sparc 20s
31 Alpha 3000/300s
13 DECstation 5000/240s
10 RS/6000s
15 SGI Indigos

Figure 1: Public clusters run by DCO

Unlike the MIT Athena environment where the typical public workstation only allows console logins, the public workstations at Stanford allow remote logins. At any one time there could be some poor user sitting at the console of a Sparc 2 with limited memory and swap space trying to read their mail while 20 other people were logged in compiling their CS project. If we had the resources we could have disabled remoted logins on all the public workstations and set up some specially configured workstations for remote logins. This is still being investigated, but for historical reasons remote logins are still allowed on public workstations. During the semester we typically saw over a thousand simultaneous unique logins across one hundred workstations.

Other Solutions

At the time I started working on lbnamed there were a number of existing solutions. "Shuffle Addresses" (SA) are one solution implemented by Bryan Beecher. One downside with SA records is they require changes to the DNS specification since they add a new resource record of type T_SA. Another solution is Marshall Rose's "Round Robin" code which is included with current versions of BIND [4]. The problem with "Shuffle Addresses" and the "Round Robin" approaches are they don't factor in load when handing out addresses. For example, the "Round Robin" code just cycles through the A records in a round-robin fashion. The "Round Robin" solution does have its benefits, as it provides some balancing with little to no expense.

At the time this paper was being written, RFC 1794 [5] was also published and describes a load balancing method using a special zone transfer agent that can obtain its information from external sources. The new zone then gets loaded by the name server. One problem with this method is in between zone transfers the weighted information is essentially static, or possibly handed out round-robin. This method also doesn't allow for exotic virtual/dynamic

domains where the response is created dynamically based on the name being queried. It does elegantly solve a class of load balancing problems though.

There have also been other load balancing name servers hacked up over the years, but most of them were like my initial lbnamed prototype and not extensible.

Requirements for Initial Implementation

The project had these initial requirements:

a. No changes to the DNS protocol, should be compatible with existing DNS implementations.
b. In between updates of load balancing information from the external source the cached load information should change so the server doesn't end up returning the same information over and over.
c. Must respond fast. Polling for load information will be done by a separate process and loaded back into the load balancing name server.
d. Should be easy to configure and maintain.
e. A host can belong to multiple groups or clusters.
f. Should not preclude having virtual/dynamic domains. The response should be dynamically generated based on the name being queried.
g. Redundancy is handled by multiple, independent servers.
h. The initial implementation is not a general purpose name server. Resolver clients should not be pointed at it and it should not be used in lieu of a real name server like BIND. Remember, don't try this at home.

Solution

lbnamed: A Load Balancing Name Server in Perl

Lbnamed is a load balancing name server written in Perl. It was meant to be a prototype that would get re-written in C and/or integrated with a special version of BIND. It has worked well enough (and I've been too busy with other things) that I've left it in Perl. Lbnamed allows you to create dynamic groups of hosts that have one name in a DNS domain. A host may be in multiple groups at the same time. For example, when someone types:

```
telnet elaine.best.stanford.edu
```

they get connected to one of 57 different SPARCstations named elaine1-elaine57. Since the Elaines contain both Sparc 2 and Sparc 20 class machines I also wanted a way for people to be able login to the "best" Sparc 2 or Sparc 20, partly for fear that people who knew the difference wouldn't want to use the "elaine.best" alias because chances are one of the Sparc 2s would have the lowest load. Therefore, someone can also type:

```
telnet sparc20.best.stanford.edu
```

or even:

```
telnet sparc2.best.stanford.edu
```

And get connected to the "best" Sparc 20 or Sparc 2.

The Server(s)

The server side consists of two Perl programs, lbnamed and poller. These programs run in parallel and communicate using signals and configuration files.

Poller

The poller daemon contacts the client daemon running on the hosts being polled. It reads a configuration file that tells the poller which hosts to poll. The poller periodically sends out requests and receives the responses asynchronously. After it has received all the responses it dumps the information into a configuration file and sends a signal to lbnamed which then reloads the configuration file. If the poller does not receive a response from one of the hosts being polled it removes it from the configuration file it feeds to lbnamed.

The poller program is also the program that calculates the weight of each system. This logic was placed in the poller program so the weight formula could easily be changed without having to modify all the poller client programs.

The formula used to determine the weight of a host is:

```
$WT_PER_USER = 100;
$USER_PER_LOAD_UNIT = 3;

$fudge = ($tot_user - $uniq_user) *
          $WT_PER_USER/5)
$weight = $uniq_user * $WT_PER_USER
      + ($USER_PER_LOAD_UNIT* $load)
      + $fudge;
```

Where the variables are:

$tot_user	total number of users logged in.
$uniq_user	unique number of users logged in.
$load	the load average over the last minute multiplied by 100.
$WT_PER_USER	the pseudo weight for each user.
$USER_PER_LOAD_UNIT	the number to multiple the load by.
$fudge	fudge factor for users logged in more then once.

The formula tries to favor hosts with the least amount of unique logins, and lower load averages. It has worked well, but could be improved.

A situation still exists when a host responds to poller requests, and has a low load, but no one can login because of a problem (such as lack of swap). A future version may be smarter and watch for trends where a host is constantly handed out but the weight of that host never changes.

The poller daemon was inspired by a previous program I wrote called fping. See the appendix for a brief description of fping.

Lbnamed

The lbnamed script reads the configuration file generated by the poller and loads it into a number of different data structures. Each group of machines is stored in an array, while the weights of all the hosts are stored in one hash table. When a request for a particular group comes in, the array for that group is sorted based on the weight of each host in that group. The host with lowest weight is then returned as the best host, and its weight is increased by adding two times the constant $WT_PER_USER to it. By increasing the weight we ensure the same host won't be returned over and over.

The best way to understand how the data is stored internally is an example. Consider the configuration file created by the poller shown in Figure 2 where the format of the file is:

```
weight host ipaddress group1 [...]
```

Upon reading the configuration file, lbnamed will create the arrays shown in Figure 3. The @group_ arrays are created using "eval":

```
eval "push(@group_$group,ost);";
```

The groups array contains all the dynamic groups and the number of members in each group. This array serves two purposes. It is used to determine if a particular group exists and to reset the current groups before the configuration file is reloaded:

```
foreach $group (%groups) {
    eval "@group_$group=();";
}
%groups=();
```

The weight array contains the weight of each host and is used to assist in sorting a particular group when a query is made. To find the host with the lowest weight, the eval function is used:

```
$the_host =
    eval "&get_best(*group_$qname);";
```

The get_best function just sorts the array passed to it using the "by_weight function:

```
sub by_weight {
    $weight{$a} <=> $weight{$b};
}
sub get_best {
  local(*group)=@_;
  local($best);
  @group = sort by_weight @group;
  $best = @group[0];
  $weight{$best} += $WT_PER_USER * 2;
  return $best;
}
```

Also note the weight of the host returned is updated so the weight does not remain static in between polls. Another option would have been to sort each array once after the configuration file has been loaded and to hand out names in a round-robin fashion until the next poll. The current method will degrade to round-robin in the case where all the hosts are equally loaded, but will tend to favor the least loaded systems in the normal case. There is room for improvement in this algorithm.

To other name servers, lbnamed looks like a standard DNS name server, with the exception that it doesn't answer recursive queries. It only handles requests for the dynamic groups it maintains. lbnamed gets a normal DNS request and, based on the name in the request, it calculates the host to return. lbnamed then constructs a standard DNS response and sends it back to client that requested it. The time to live (TTL) value in the response is set to 0. This prevents the response from being cached by other name servers.

```
1364 elaine1 36.215.0.117 elaine sparc2 sparc sunos
1264 elaine2 36.215.0.118 elaine sparc2 sparc sunos
1602 elaine40 36.218.0.88 elaine sparc20 sparc sunos
1827 elaine41 36.218.0.89 elaine sparc20 sparc sunos
```

Figure 2: Poller configuration file

```
@group_sparc20 = ( "elaine40","elaine41");
@group_sparc2  = ( "elaine1", "elaine2");
@group_elaine = ( "elaine1", "elaine2", "elaine40","elaine41");
@group_sparc = ( "elaine1", "elaine2", "elaine40","elaine41");
@group_sunos = ( "elaine1", "elaine2", "elaine40","elaine41");

%groups = ('sparc20',2,'sparc2',2,'elaine',4,'sparc',4,'sunos',4);

%weight = ('elaine1',1364,'elaine2',1264,'elaine40',1602,'elaine41',1827);
```

Figure 3: Arrays created from configuration file

For example, Figure 4 shows the use of "dig" (which is distributed with the latest version of BIND[4]) to see data returned from a query to the load balancing name server. Figure 5 shows a second query for the same domain with a different returned value.

There are a few things to note in the data returned. First, a dynamic CNAME is returned, not a dynamic A record. By returning a dynamic CNAME we can leverage off other data associated with the real domain name (such as an MX record). In addition, by returning a CNAME the resolver client doesn't end up with an A record that doesn't have a corresponding PTR record.

Secondly, note that we have returned the address of the host to which the CNAME points. This should save an extra lookup by the resolver client. If we just returned the CNAME record, the client would then have to lookup the A record for that name as well. We already have the IP address because the poller needs it to poll the host.

Load Balancing Client Daemon

Hosts that are going to be polled by the poller need to run a special daemon, the load balancing client daemon (lbcd). lbcd responds to poller requests (over UDP) using a simple protocol. The protocol format is described in the appendix. I wrote yet-another-remote-statistics daemon because initially I had grand plans for having it do a number of different system management tasks. Later, I used Sysctl [6] for those tasks.

lbcd is written in C, although it probably could have been written in Perl with a few helper programs written in C to read information from the kernel. The important thing to remember is the client and poller can easily be replaced with something else, as long as the poller program creates the lbnamed configuration file in the correct format.

Configuring the load balancing name servers

For the load balancing name server to answer requests it must be delegated a virtual domain to serve. This is normally done in the parent domain by adding NS records. In the stanford.edu domain the load balancing name server uses the best.stanford.edu domain, so in the DNS configuration file for the stanford.edu domain there are two NS records:

```
best   IN   NS dsodb.stanford.edu.
best   IN   NS sunlight.stanford.edu.
```

These two NS records delegate the best.stanford.edu domain to the lbnamed's running on dsodb and sunlight. Now when the primary servers for the stanford.edu domain get a request like elaine.best.stanford.edu they know to forward it to the lbnamed's. Note that the two lbnamed's don't communicate with each other; they both operate independently for simplicity and redundancy.

```
# dig elaine.best.stanford.edu

; <<>> DiG 2.1 <<>> elaine.best.stanford.edu
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 6
;; flags: qr aa rd ra; Ques: 1, Ans: 2, Auth: 0, Addit: 0
;; QUESTIONS:
;;    elaine.best.stanford.edu, type = A, class = IN

;; ANSWERS:
elaine.best.stanford.edu.      0     CNAME elaine19.stanford.edu.
elaine19.stanford.edu.  3600  A     36.216.0.207

;; Total query time: 60 msec
;; FROM: cardinal1.Stanford.EDU to SERVER: default -- 36.56.0.150
;; WHEN: Thu Jul  6 22:13:57 1995
;; MSG SIZE  sent: 42  rcvd: 114
```

Figure 4: Dig results of first query

```
# dig elaine.best.stanford.edu

 [... header deleted ...]

;; ANSWERS:
elaine.best.stanford.edu.      0     CNAME elaine16.stanford.edu.
elaine16.stanford.edu.  3600  A     36.216.0.204
```

Figure 5: Dig results from second query

Perl 5 Server

Although the Perl 4 version has worked fine, it serves a single purpose: handing out load information in the "best.stanford.edu" domain. It is possible to modify it to serve other information was well, but doing so using Perl 4 would not have been easy due to the lack of nested data structures. After this paper was accepted, I decided to re-write the name server in Perl 5, using Perl 5 features like references to achieve nested data structures, and the Socket module to provide portability.

Server Organization

The Perl 5 server is organized into four different files: DNS.pm, lbnamed, lbnamed.conf, and LBDB.pm.

DNS.pm

DNS.pm is a Perl 5 package containing constants and functions that assist in creating and parsing DNS messages. It was created by starting with /usr/include/arpa/nameser.h and converting it into Perl 5. From there functions were added to expand compressed domain names, create resource records, etc. After importing DNS.pm, programs can use these functions and constants. For example:

```
$flags |= QR_MASK | AA_MASK | NOTIMP;
```

Functions are provided for encoding data into resource records (RRs). After the RR is encoded it is returned as string. Only the common RRs were implemented, see Figure 6.

An answer to a DNS query consists of of 6 pieces of data. The domain name which the resource records pertains to, the type of the RR, the class of the RR, the time-to-live (TTL) of the data, the length of the data, and the data the itself. The dns_answer function is used to create an answer:

```
$answer = dns_answer(QPTR, T_TXT,
    C_IN, 60, rr_TXT($date));
```

Note that the length of the resource data is not passed to the dns_answer function because the resource data is passed in as a string. The dns_answer function uses the string's length as the resource data's length since Perl strings can contain null characters, unlike null-terminated C strings. Also note the special constant "QPTR". QPTR is a compressed domain name pointer which points to the original question in the DNS message. If the domain name of RR that is getting added to the answer section is the same as the domain name in the question you should use QPTR. QPTR takes only two bytes as opposed to duplicating the original domain name.

lbnamed

lbnamed is the main server. It loads lbnamed.conf, sets up the TCP and UDP sockets, and then answers requests after performing a select(2) on the TCP and UDP sockets. Upon receiving a request it calls the do_dns_request function which attempts to parse the DNS request. If the request is invalid (i.e., a unsupported operation is requested or the request could not be parsed), an error is returned. Otherwise lbnamed first checks to see if there is a static answer available. If not, it attempts to find a dynamic domain that will answer the question. If neither a static or dynamic answer is found then the NXDOMAIN (non-existent domain) error is returned.

A static domain name is a domain name that does not change from one query to the next. A dynamic domain name is a domain that can possibly change from query to query. Static and dynamic domains are discussed in detail a little later.

Figure 7 shows the code for checking for static and dynamic domain names. The response to the query is generated using Perl's "pack" function, as shown in Figure 8. Note that the LBDB::check_static and LBDB::check_dynamic functions are free to modify the various variables in the $dnsmsg associative array, such as setting the response code (rcode) and adding data to the answer, authority, and additional section of the response.

```
$data = rr_A($ipaddress);
$data = rr_CNAME("stanford.edu");
$data = rr_HINFO("PowerPC","Solaris/2.5");
$data = rr_MX(10,"leland.stanford.edu");
$data = rr_NS("bestserver.stanford.edu");
$data = rr_NULL;
$data = rr_PTR("leland.stanford.edu");
$data = rr_SOA("foo.stanford.edu","root.stanford.edu",
            1234, 1200, 300, 604800,86400)
$data = rr_TXT("this is text");
```

Figure 6: Prototypes for implemented common resource record encodings

```
if (LBDB::check_static($qname,
      $qtype,$qclass,$dnsmsg)) {
   # return answer
} elsif (LBDB::check_dynamic($qname,
      $qtype,$qclass,$dnsmsg)) {
   # return answer
} else {
   $dnsmsg->{'rcode'} = NXDOMAIN;
}
```

Figure 7: Coding to check static/dynamic domain names

lbnamed.conf

lbnamed.conf is the place to put local modifications, and to define two function hooks which are called from lbnamed: do_maint, clean_exit. The do_maint function is called from the answer_requests function in lbnamed if the variable need_maint is set. For example, lbnamed.conf can install a signal handler to catch the HUP signal. This signal handler would set the need_maint variable so the do_maint function would get called. clean_exit is a function which cleanly shuts down the server.

lbnamed.conf also contains calls to the LBDB::add_static and LBDB:add_dynamic functions to add static and dynamic data. Under normal circumstances lbnamed.conf is the only file that needs to be changed.

LBDB.pm

LBDB.pm is a Perl 5 package that contains the functions for adding data to and checking for static and dynamic domain data.

Registering Static Domains

Static domain data (data that does not vary from query to query) is added using the LBDB::add_static function as shown in Figure 9. The database for static information is implemented using a four-level hash table:

```
$static_domain{$domain} ->
    {$dns_class} -> {$dns_type} =
          { ...data... }
```

The first-level hash table is indexed by the domain name of the data, the second level is indexed using the class of the data (such as C_IN), the third level is indexed using the type of the data (such as T_A), and the fourth level contains the information associated with the data of that domain, class, and type. This layout simplifies finding all the data associated with a given domain name, even when confronted with a query that contains C_ANY or T_ANY.

Static domain data can be used for any type of data, but will probably be used mainly for answering SOA queries for a dynamic domain. For example, to register an SOA record for the "best.stanford.edu" domain you would make the following call in the "lbnamed.conf" file:

```
LBDB::add_static("best.stanford.edu",
    T_SOA,
    rr_SOA(hostname, $hostmaster,
       time, 86400, 86400, 86400, 0)
);
```

```
$flags |=  QR_MASK | AA_MASK | $dnsmsg->{'rcode'};
$response = pack("n n n n n n", $id, $flags, $qdcount,
    $dnsmsg->{'ancount'}, $dnsmsg->{'nscount'}, $dnsmsg->{'arcount'})
        . $question
        . $dnsmsg->{'answer'}
        . $dnsmsg->{'auth'}
        . $dnsmsg->{'add'};
```

Figure 8: Creating response to query

```
sub add_static {
  my($domain,$type,$value,$ancount,$class,$ttl) = @_;

  $ancount = 1            unless $ancount;
  $class   = C_IN         unless $class;
  $ttl     = $default_ttl unless $ttl;

  $static_domain{$domain} -> {$class} -> {$type} = {
            "answer" => dns_answer(QPTR,$type,$class,$ttl,$value),
            "ancount" => $ancount
  };
}
```

Figure 9: LBDB::add_static

Registering Dynamic Domains

Dynamic domains (data that gets created dynamically, based on the name being queried) are added using the LBDB::add_dynamic function as shown in Figure 10. The database for dynamic information is implemented using a hash table:

```
$dynamic_domain{$domain} = $handler;
```

The hash table is indexed by the domain name of the data, and the value returned is a reference to a function which gets called at the time the query is made. The function for a dynamic domain is called with the following arguments:

```
&$dfunc($domain, $residual, $qtype,
    $qclass, $dnsmsg);
```

$domain is the dynamic domain (i.e., best.stanford.edu)
$residual is the data to the left of the dynamic domain (i.e., elaine)
$qtype is the type of the of the query (i.e., T_A)
$qclass is the class of the query (i.e., C_IN)
$dnsmsg is a reference to a hash table which is used to return information to the load balancing name server.

The function returns 1 if it executed successfully (i.e., the results in $dnsmsg should be used) or 0 otherwise.

```
sub add_dynamic {
  my($domain, $handler) = @_;
  $dynamic_domain{$domain} = $handler;
}
```

Figure 10: LBDB::add_dynamic

The algorithm for finding a dynamic domain attempts to find the longest dynamic domain name that matches the query. For example, if we had the following dynamic domains registered:

```
stanford.edu
best.stanford.edu
```

And the following query came in:

```
elaine.best.stanford.edu
```

Then the handler for the "best.stanford.edu" domain would be called, since it is the longest match for "elaine.best.stanford.edu" Here is how the algorithm matches "elaine.best.stanford.edu":

```
    domain                 residual  match
"elaine.best.stanford.edu"   ""        no
"best.stanford.edu"        "elaine"   yes
```

If the query was "foo.bar.stanford.edu" then the match would look like:

```
    domain              residual  match
"foo.bar.stanford.edu"   ""        no
"bar.stanford.edu"      "foo"      no
"stanford.edu"        "foo.bar"    yes
```

Dynamic domains are the heart of the load balancing name server as they allow you to create answers dynamically based upon the name being queried. The best way to explain dynamic domains is with an example.

Let's create a domain called "random.stanford.edu", which will return a different random number between 0 and 10 every time it is called. We register that domain by adding the following calls to lbnamed.conf:

```
sub handle_random {
   my($domain, $residual, $qtype,
          $qclass, $dm) = @_;
   $dm->{'answer'} .= dns_answer (
        QPTR, T_TXT, C_IN,
        60, rr_TXT(int(rand(10))));
   $dm->{'ancount'} += 1;
   return 1;
}
LBDB::add_dynamic(
    "random.stanford.edu" =>
    \&handle_random);
```

By calling LBDB::add_dynamic we are requesting that the load balancing name server call our function whenever a request comes in for the name "random.stanford.edu". The first statement calls the dns_answer function which creates the binary data which will be placed in the answer section of the DNS message:

```
$dm->{'answer'} .= dns_answer(
         QPTR,  T_TXT, C_IN,
         60, rr_TXT(int(rand(10)))
);
```

QPTR is a constant defined in DNS.pm that is a compressed domain name pointer which points to the original question in the DNS message. T_TXT is the type of data being returned. C_IN is the class of the data being returned. 60 is the time-to-live (TTL) of the data being returned (in seconds), and rr_TXT is a function which given a text string returns a text resource record. The second statement increments the answer count in the reply message. The response code for the reply is set to NOERROR by default, so there is nothing else for us to set.

Let's say we also want to define a domain that returned a random number between 0 and 100. It would be easy to do something like:

```
LBDB::add_dynamic(
    "random100.stanford.edu"
      => \&handle_random_100);
```

But that solution does not scale. The solution is to modify the original handle_random function so that it examines the residual part of the domain name passed to it. For example:

```
sub handle_random {
   my($domain, $residual, $qtype,
              $qclass, $dm) = @_;
   $residual = 10 unless $residual;
   $dm->{'answer'} .= dns_answer(
       QPTR, T_TXT, C_IN, 60,
       rr_TXT(int(rand($residual))));
   $dm->{'ancount'} += 1;
   return 1;
}
```

This enables us to make a query in the form:

```
random.stanford.edu
```

or:

```
N.random.stanford.edu
```

where the return value will be between 0 and N; see Figure 11 for an example. While "random.stanford.edu" is not a useful domain, it helps show the basic concepts involved in creating dynamic domains. A more useful example would create a dynamic domain that mimicked the Hesiod "passwd" domain in the Athena environment.

Using a standard BIND server that understands class HS and type TXT records, you need a database entry in the domain file for each user in the password file; see Figure 12 for an example. If you have a large password file (like Stanford's 22,000 users), then BIND will consume a lot of memory loading every single passwd entry. It also means that to add/delete/update an entry you'll have to reload the whole file. Using lbnamed you register a dynamic domain:

```
sub handle_passwd_request {
  my($domain, $residual, $qtype,
                $qclass, $dm) = @_;
  my($name, $passwd, $uid, $gid,
     $q, $c, $gcos, $dir, $shell) =
      getpwnam($residual);
  my($entry);
  if ($name) {
     $entry = "$name:*:$uid:$gid:".
                "$gcos:$dir:$shell";
  } else {
       $dm->{'rcode'} = NXDOMAIN;
       return 1;
  }
  $dm->{'answer'} .=  dns_answer(QPTR,
      T_TXT, C_HS, 3600,
      rr_TXT($entry));
  $dm->{'ancount'} += 1;
  return 1;
}

LBDB::add_dynamic(
    "passwd.ns.stanford.edu"  =>
          \&handle_passwd_request);
```

Now if someone attempts to lookup the name "root.passwd.ns.stanford.edu", the lookup will get re-directed to the handle_passwd_request, which will lookup the passwd information and construct the correct response dynamically. Note that depending on the OS, the getpwnam call could be getting the password information from a local file, a DBM file, or even NIS/NIS+. You could also replace the getpwnam call with your own function that obtains information from a DBM file or even a relational database.

Results/Conclusions

Overall lbnamed has been a big win at Stanford. It has helped distribute the load among a large number of workstations. It also enables system administrators to take systems down temporarily without interrupting users who use the load balancing name. It also allows systems to be transparently added and removed from groups.

The problems have been minor. The biggest problem has been hosts that respond to load balancing queries, but don't allow logins (due to other problems). This could be fixed by falling back to a strict round-robin scheme in between polls, or some variant, such as changing the weight of the host handed out to be slightly more than the host in the middle of the list, for example.

```
# dig 100.random.stanford.edu

 [... header deleted ...]

;; ANSWERS:
100.random.stanford.edu.       60     TXT    "97"

 [... trailer deleted ...]
```

Figure 11: Querying N.random.stanford.edu

```
root.passwd HS TXT "root:*:0:1:Root Account:/:/bin/sh"
```

Figure 12: Sample database entry for root user

The other problem has been resolver clients that don't deal well with TTL values of 0. This has only happened in a few cases and generally only happens in clients with old software.

Some people may not feel comfortable using a TTL of 0, but I personally don't have any trouble sleeping at night because I chose it. As mentioned in RFC 1794, there are plenty of versions of BIND that treat anything less than 300 seconds as 300 seconds, which can defeat the whole purpose of trying to balance the load. I figured the added load on the name servers was worth the benefit of getting a truly dynamic response to every query. When trying to load balance your cache can be trash...

Future Directions

One of the reasons I decided to write this paper was to get people thinking about "exotic" name servers. There are a number of directions someone could take the concepts presented here, some of which have already been hinted at in a future version of BIND. For example, the registration of dynamic domains could be added to BIND by loading shared objects at runtime, or by allowing external daemons to register with BIND and communicate via IPC mechanisms.

As far as the Perl implementation of lbnamed is concerned, a number of improvements immediately come to mind:

- recognizing when a particular host is consistently handed out as being the best, even though no one can login to it.
- adding more factors to the determine the weight of a host, such as a swap space, free memory, number of processes, CPU model, etc.
- modifying the poller protocol so poller clients can specify their weight rather then letting the poller calculate it for them. For example, you could load balance requests to a name such as "www.stanford.edu" based on the average number of requests over the last few minutes, etc.
- modifying the poller so it periodically reloads the IP addresses of clients into its cache.
- adding logging and statistics back to the Perl 5 version.
- generalize support for domain name compression in answers.

Acks

The Perl 4 version was written while I was at Stanford. The Perl 5 code was written (in my free time after work) after I left Stanford, and after this paper was accepted. Special thanks to Shirley Gruber at Stanford University and Kevin Kluge at SunSoft for finding and correcting plenty of errors in early versions of this paper. My English is a little better and they probably know a little more about DNS ;-)

Availability

The code is available using the following URL: http://www-leland.stanford.edu/~schemers/dist/lb.tar Use the code at your own risk. The Perl 4 version has been in use at Stanford for over two years.

Author Info

Roland Schemers received his M.S degree in Computer Science from Oakland University in Rochester, Michigan. He currently is working in the DCE engineering group at SunSoft. He previously worked in the Distributed Computing Group at Stanford, and helped manage and maintain such campus-wide services as AFS, Kerberos, and DNS, as well as the public workstation clusters and servers. He can be reached electronically at <schemers@eng.sun.com>.

While at Stanford he also co-authored a chapter in the book *Distributed Computing, Implementation and Management Strategies*, Raman Khanna, Editor, which probably would have sold better if it had the words "Client/Server" in the title.

He is patiently waiting for the day when he can login into any UNIX system and access /usr/bin/perl.

References

[1] Larry Wall, Randal L. Schwartz, *Programming perl*, O'Reilly and Associates, Sebastopol, CA.

[2] Stephen P. Dyer, *The Hesiod Name Server*, Proc. USENIX Winter Conference, 1988.

[3] Paul Albitz, Cricket Lui, *DNS and BIND*, O'Reilly and Associates, Sebastopol, CA.

[4] Paul Vixie, BIND, http://www.isc.org/isc/

[5] Thomas P. Brisco, *DNS Support for Load Balancing*, RFC 1794.

[6] Salvatore DeSimone, Christine Lombardi, *Sysctl: A Distributed System Control Package*, Proc. USENIX LISA Conference, November 1993.

[7] Dan Farmer, Wietse Venema, SATAN, satan@fish.com.

APPENDIX

Poller configuration file

The poller configuration file tells which hosts the poller should poll, and which dynamic groups those hosts are in. The format is:

```
host    weight-multiplier   group1 [group2 ...]
```

The weight-multiplier field is currently not used but could be used in the future to allow for better selection among different hardware in the same group.

The following is a sample poller configuration file with some lines removed to save space.

```
#
# groups
# ------------------------
# sweet     all machines
# elaine    elaine1-elaine57
# sparc     elaine1-elaine57
# sunos     elaine1-elaine57
# sparc2    sparc2 (elaine1-elaine19)
# sparc1    sparc1 (elaine20-elaine57)
# adelbert  adelbert1-adelbert26
# ultrix    adelbert1-adelbert26
# dec       adelbert1-adelbert26
# dec5000   adelbert1-adelbert13
# dec3100   adelbert14-adelbert26
# rs        rs1-rs10
# rs6000    rs1-rs10
# aix       rs1-rs10
#
rs1    1    rs rs6000 aix
rs2    1    rs rs6000 aix
rs10   1    rs rs6000 aix
#
elaine1    1    elaine sparc2 sparc sunos sweet
elaine2    1    elaine sparc2 sparc sunos sweet
elaine19   1    elaine sparc2 sparc sunos sweet
#
elaine20   1    elaine sparc1 sparc sunos sweet
elaine21   1    elaine sparc1 sparc sunos sweet
elaine57   1    elaine sparc1 sparc sunos sweet
#
adelbert1  1    adelbert dec5000 dec ultrix sweet
adelbert2  1    adelbert dec5000 dec ultrix sweet
adelbert13  1    adelbert dec5000 dec ultrix sweet
#
adelbert14  1    adelbert dec3100 dec ultrix sweet
adelbert26  1    adelbert dec3100 dec ultrix sweet
#
```

lbnamed configuration file

The lbnamed configuration file tells lbnamed what the weight of each host is, what its IP address is, and which dynamic groups a hosts is in. The format is:

```
weight host ipaddress group1 [group2 ...]
```

The following is a sample lbnamed configuration file with some lines removed to save space.

```
2200 elaine11 36.214.0.127 elaine sparc2 sparc sunos sweet
639 adelbert10 36.211.0.81 adelbert dec5000 dec ultrix sweet
651 elaine20 36.215.0.208 elaine sparc1 sparc sunos sweet
2336 elaine3 36.212.0.119 elaine sparc2 sparc sunos sweet
...
```

```
866 adelbert6 36.211.0.76 adelbert dec5000 dec ultrix sweet
243 adelbert26 36.212.0.201 adelbert dec3100 dec ultrix sweet
```

Protocol

The protocol between the poller and client daemon is simple. Everything is in network byte order. I used UDP so I could easily send out multiple polls at the same time and receive responses asynchronously. The packet format (described by C structures) is:

```
#define PROTO_PORTNUM 4330
#define PROTO_MAXMESG 2048     /* max udp message to receive */
#define PROTO_VERSION 2

typedef enum P_OPS {
    op_lb_info_req              =1,  /* load balance info, request and reply */
} p_ops_t;

typedef enum P_STATUS {
    status_request              =0,  /* a request packet */
    status_ok                   =1,  /* ok */
    status_error                =2,  /* generic error */
    status_proto_version        =3,  /* protocol version error */
    status_proto_error          =4,  /* any other protocol error */
    status_unknown_op           =5,  /* unknown operation requested */
} p_status_t;

typedef struct {
  u_short   version;  /* protocol version */
  u_short   id;       /* requestor's uniq request id */
  u_short   op;       /* operation requested */
  u_short   status;   /* set on reply */
} P_HEADER,*P_HEADER_PTR;

typedef struct {
  P_HEADER h;
  u_int boot_time;
  u_int current_time;
  u_int user_mtime;  /* time user information last changed */
  u_short l1; /* (int) (load*100) */
  u_short l5;
  u_short l15;
  u_short tot_users;  /* total number of users logged in */
  u_short uniq_users; /* total number of uniq users */
  u_char  on_console; /* true if someone on console */
  u_char  reserved;   /* future use, padding... */
} P_LB_RESPONSE, *P_LB_RESPONSE_PTR;
```

The protocol was meant to be extensible but I have yet to use the daemon for anything but load balancing requests.

fping

The poller daemon was inspired by a previous program I wrote called fping. fping is a ping(8) like program which uses the Internet Control Message Protocol (ICMP) echo request to determine if a host is up. fping is different from ping in that you can specify any number of hosts on the command line, or specify a file containing the lists of hosts to ping. Instead of trying one host until it times out or replies, fping will send out a ping packet and move on to the next host in a round-robin fashion. If a host replies, it is noted and removed from the list of hosts to check. If a host does not respond within a certain time limit and/or retry limit it will be considered unreachable. fping is used by SATAN [7] to quickly ping a list of hosts and/or ip addresses.

fping is currently being maintained and updated by R. L. "Bob" Morgan <morgan@networking.stanford.edu> and can be obtained via the following URL:

```
ftp://networking.stanford.edu/pub/fping/fping.2.0.tar.gz
```

LPRng – An Enhanced Printer Spooler System

Patrick Powell – San Diego State University, San Diego, CA
Justin Mason – Iona Technologies, Ireland

ABSTRACT

The LPRng software is an enhanced, extended, and portable version of the Berkeley LPR software. While providing the same general functionality, the implementation is completely new and provides support for the following features: lightweight (no databases needed) lpr, lpc, and lprm programs; dynamic redirection of print queues; automatic job holding; highly verbose diagnostics; multiple printers serving a single queue; client programs do not need to run SUID root; greatly enhanced security checks; and a greatly improved permission and authorization mechanism.

Introduction

Print spooler software is one of the most common and heavily used system application programs. While printing may appear to be simple on the surface, in practice it is complicated by the following problems. Each model of printer has a peculiar set of interface and format requirements; this means that the printer software must be highly configurable at the device interface level. Next, multiple users may want to share the same printer; this leads to the need for a spooling system with the associated problems of priority and fair use. Printers are notorious for failing at the most inopportune times; the spooling software needs to report failures and to reconfigure or repair the system in a simple manner. Finally, the software should be portable so that the same software can be used on different systems; in a network based system this introduces the problems of security and authentication.

The LPRng Printer Spooling [Pow95] software is a descendant of the 4.3 BSD Line Printer Spooler Software (LPR), [Cam94] but has totally redesigned and reimplemented. The evolution started in 1986 at the University of Waterloo, where the original 4.3 software was modified to support a variety of new printers. Due to restrictions with the original AT&T and Berkeley software license these modifications could not be distributed. The problems encountered during this process led to the development of the PLP (Public Line Printer) software [Pow95a] and PLP Version 3.0 (PLP3.0) was released in 1988. The PLP software architecture was based on the the original LPR code, but with highly verbose diagnostics and a much more elaborate set of administration functions.

From 1988 to 1994 various sites and administrators modified and extended the PLP3.0 software. The `plp@iona.ie` mailing list was formed to distribute and coordinate these changes, and in 1994 a major programming effort by Justin Mason <jmason@iona.ie> restructured the PLP3.0 code, integrated the majority of extensions, and PLP4.0 was released in 1995.

The complexity and problems with the PLP software have been discussed in the `plp@iona.ie` mailing list as well as various USENIX newsgroups. Given the current network security issues, client/server based applications, and growing administration problems, it was clear that the PLP4.0 software would need extensive revisions. The need for a new version of print spooling software discussed, and there was general agreement on the following design goals.

First, run time diagnostics and detailed error reporting were essential and should be the highest priority. When problems occur users and administrators must quickly diagnose the causes, and obtaining information is essential. Next, the user interface to the printing facilities should change as little as possible. This would allow a gradual evolution from LPR and PLP to the new software with as least surprises to the users as possible. However, the administrative interface could change, and many improvements and changes were suggested. It was essential that the new software be compatible at the network interface level with other implementations of the LPR spooling software. While in 1990 the RFC1179 – Line Printer Daemon Protocol [McL90] documenting the network protocol to be used to transfer print jobs and status information between line printer spooling programs was published, many of the existing implementations do not conform to RFC1179 or have made extensions to the RFC. The existing LPR and PLP software uses a set of *filter* programs to interface to various printers. A major concern of administrators was that these *vintage* filter programs should be usable with the new software. Finally, the long list of security, administration, and networking problems should be eliminated if at all possible.

These considerations led to the design and development of the LPRng software. While it is a totally new design and implementation of spooling software, it uses routines and support code from the Free Software Foundation GNU Project, and is distributed under the GNU Copyleft License. [GNU91] The LPRng software was intentionally designed to use as few non-portable or non-standard Operating System facilities as possible, or to use them in a highly controlled and portable manner. The use of the GNU utilities such as *autoconf* and *Gnumake* allow operating system dependent versions of various support routines to be selected at compile time in an automatic manner.

The following sections discuss the overall architecture of the LPRng software, and then deal with the major components. The emphasis of this discussion are the added functionality or differences of LPRng. The LPRng configuration information, extensions to the printcap database, and changes to the `lpr` and other client programs is discussed. The operation of the job spool queues and the new algorithm used for job printing is then covered, together with a description of the filter interface mechanism. Security and associated problems with SETUID ROOT programs is briefly discussed, and the summary at the end lists some outstanding issues.

LPRng Software Architecture

The LPRng software architecture is shown in Figure 1. While LPRng is similar in structure to the Berkeley LPR software, it differs in many important details. The dashed lines indicated TCP/IP based communication between two programs; solid lines represent access to files or directories. Boxes with dotted outlines represent databases that may be accessed by all programs, either as files or by using network facilities. The user programs such as the print spooler `lpr`, status reporter `lpq`, job remover `lprm`, and control program `lpc` are now client programs which connect to one or more `lpd` server processes using TCP/IP. After validation and authentication the servers carry out requested activities on files and or provide status information. The *configuration* and *printcap* databases provide the information needed by both server and client programs. While clients do not need access to the printcap database, in many cases a *runt* database is useful for providing printer configuration information.

As in the LPR software, the `lpd` server manages one or more spool queues where print jobs are stored. These are are implemented as directories in a file system. A print job consists of a control file, which contains user information and printing options, and data files which contain the actual information to be printed. A spool queue can be a *bounce* or *forwarding* queue, which temporarily stores print jobs before they are transferred to another queue, or a *print* queue which has an associated printer.

Operation of a spool queues is controlled by information in the spool queue printcap entry and the printer control file in the spool directory; individual print job may also have a job control file as well.

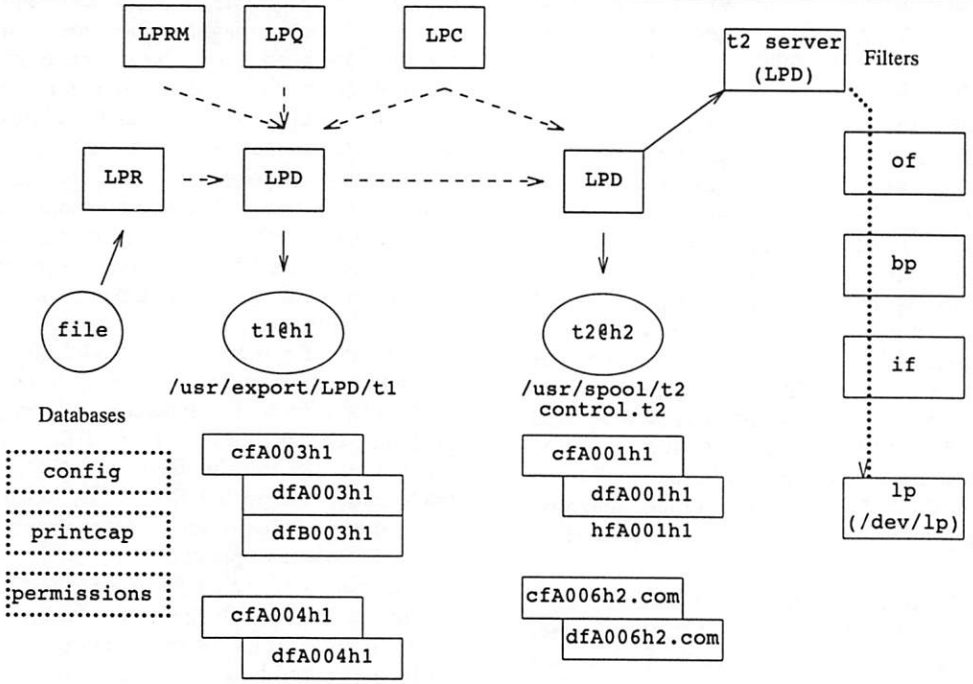


Figure 1: LPRng Spooling Software Architecture

Jobs are submitted to the lpd server by the lpr program which transfers the job over a TCP/IP connection. The lpd server then forwards the job to another server or print it. The lpq program requests and prints job status information, and the lprm program removes jobs from the spool queue. The LPRng software uses a *permissions* database and the printcap information to determine if a user is authorized to use a facility; authorization can be based on originating host, user name, and a variety of other attributes.

After a job is placed in a print queue, lpd creates a server process to manage the printing operations. This server process then creates the necessary filter processes which interface to the printer hardware. The data files are passed through the filters to the actual printer.

Configuration Information

Configuration information is used by both the LPRng clients and the lpd server. The configuration information controls the network behavior of the programs, and provides a set of default for commonly specified system information. Compile time defaults can be overridden by values read from a configuration file, whose format is shown in Figure 2.

In all LPRng database files leading whitespace, blank lines, and lines whose first non-whitespace character is a # are treated as comments and ignored; a \ as the last character of a non-comment line will logically continue this line to the next line, replacing the \ with one or more spaces.

Each line of the configuration file has a configuration variable and its value. The client_configuration_file and server_configuration_file values are used only at startup and initialization, and specify the configuration files for the LPRng client and lpd server programs. Each of the configuration files is read in sequence and variable values are updated as the files are read.

Much of the configuration information provides site dependent information or allows configuration for testing. The default_printer and default_host variables set the default printer and host to be used by client software; the %h and %H strings are replaced with the short or fully qualified domain name of the host on which the software is running. The default_banner_printer sets the default banner printing program to be used by the lpd server; the lockfile and logfile are used by the lpd server to prevent multiple servers from running and to record lpd logging information.

The lpd_port variable specifies the TCP/IP port on which the lpd server listens for client requests. In production versions this is usually 515 (the printer alias in the network service database); by setting it to some other port a test version can be run in parallel with production software.

```
# ENG LPRng Test Configuration
# compile time only:
#client_configuration_file /etc/lpd.conf:/etc/lpd_client.conf
#server_configuration_file /etc/lpd.conf

default_printer          t1
default_host             %H
default_banner_printer   /usr/local/bin/lpbanner
lockfile                 /usr/spool/LPD/lpd.lock
logfile                  /usr/adm/lpd.log
#lpd_port                printer
lpd_port                 4000
originate_port           721 731
user                     daemon
group                    daemon
#printcap_path           /etc/printcap:/usr/etc/printcap
printcap_path            /tmp/LPD/printcap.%H
#printcap_path           |/tmp/LPD/pcserver
#printer_perms_path      /tmp/LPD/printer_perms.%H
#printer_perms_path      /etc/printperm:/usr/etc/printperm
printer_perms_path       /tmp/LPD/printer_perms.%H
#print_perms_path        |/tmp/LPD/permserver
use_info_cache           yes
# include facility
include                  /tmp/LPD/common.conf
```

Figure 2: Configuration Database Format

The originate_port value specifies a range of TCP/IP port numbers for originating connections. RFC1179 specifies that these connections should originate from port 721 to 731 inclusive; in most UNIX environments these are *privileged* ports and cannot be used unless the program's effective UID is ROOT (0). On a UNIX system, if the client software is not SETUID ROOT, then only the ROOT user can successfully bind to a privileged port. See Security Considerations for details on problems this may expose. The user and group entries specify the effective user and group IDs to be used by the lpd server. For this to be effective, the lpd server must be SUID root or be started by a root process; see Security Considerations for details.

The printcap_path and printer_perms specify where database information will be found; this information may need to be read repeatedly by the lpd server. The use_info_cache option allows the server to read the information once at startup and then use a cached copy of this information, as does the inet.d server. If lpd receives a SIGHUP signal it rereads the database information. Finally, it is possible to use the include facility to read additional configuration files. This facility may be removed in later releases of the LPRng software.

Printcap Information

Entries in the printcap database define spool queues and their configuration available to the LPRng software. Figures 3a and 3b show a set of client and server printcap database entries. Leading whitespace, blank lines, and lines whose first character is '#' are ignored. For compatibility with the historical LPR printcap format, \ at the end of a line appends the next line to the current line.

```
# Client Printcap Database
# printer p1@'local host'
p1
# remote printer
p2
  |full|double|rotate
  |twosided|XDR Line Printer
  :lp=p2@host
# remote printer alternative
p3:rp=p3:rm=host
# connect to port 2000
p4:2000%host
# all entry (lpq -a)
all:all=p1,p2,p3
```

Figure 3a: Client Printcap Examples

A printcap entry consists of a primary name followed by an optional set of aliases, followed by an optional set of variable tag names and values. The primary name is the name by which the printer is referred to in error messages and status information. The | separator starts an alias entry and the : separator starts an variable entry; entries extend to the end of line or the next separator character; leading and trailing in each entry whitespace is ignored.

The LPRng client programs need only the lpd server host name and target printer on the server. This can be specified on the command line using the '-Pprinter' or '-Pprinter@host' option; if no default is specified in the configuration information the local host is the default server host. In Figure 3a, the simple printcap entry p1 means printer p1 on the default host; entry p2 has has two aliases, the last of which is really a comment and will be used when displaying status information.

```
# Server/Client Printcap Database
# file: /etc/printcap
# clients see p1 as remote pr
# server use sd tag to get
#  /usr/spool/LPD/p1/printcap
p1
  :cm=Test Printer 1
  :sd=/usr/spool/LPD/p1
  :lp=p1@host
# second printer,
p2
  :sd=/usr/spool/LPD/p2
  :tc=common
# common information
common:
  :lf=log
  :rw
  :of=/tmp/LPD/psof
  :if=/tmp/LPD/psif

# Printer specific information
# used by server,
# file: /usr/spool/LPD/p1/printcap
p1
  # override previous value
  :lp=/dev/ttya
  :lf=log
  :rw
  :of=/tmp/LPD/psof
  :if=/tmp/LPD/psif
# debug
#   :db=9,remote=10
# autohold
#   :ah
```

Figure 3b: Server/Client Printcap Examples

The lp (line printer) tag specifies the printer device or host. The form lp=printer@host is printer on host; the form lp=printer@host%2000 indicates the lpd server is available on port 2000. This last form is useful when running multiple versions of spooler software and when connecting to speical network based printers. A file pathname such as lp =/dev/ttya specifies a printer device to be used

by the server; the form `lp=host%2000` indicates port 2000 on host is network based printing device.

More printcap information is needed for the `lpd` server, as is shown in Figure 3b. Spool queues have printcap entries with a `sd` (spool directory) tag. The `tc` tag (recursively) appends a printcap entry to the end of the referencing entry.

The `lpd` server checks to see if a `printcap` file is in the spool directory, and will read the printcap information from this file, overridding existing information. This allows a single *master* printcap database to be used by both clients and servers; the clients ignore the `sd` tags and the server gets printer specific information from the printcap file in the spool directory.

A major administration problem is the distribution of printcap information. One solution is to use a network database such as Sun Microsystems NIS, HESIOD, Sybase, etc. Rather than build in a specific database access method the LPRng software uses the concept of *database filters* to access the information. In Figure 2, the configuration `printcap_path` value `|/tmp/LPD/dbserver` specifies using a filter program to get printcap information.

The filter program is started by the client or server process and a string containing the name of the desired printcap entry is sent to the filter's `stdin` port; the returned printcap information is read from the filter's `stdout` port. By convention, a `all` request returns either all the available printcap entries, or an `all` printcap entry whose `all` tag contains a list of available printers.

The Sun NIS database can be access by using a simple shell script and the `ypmatch` program; HESIOD, DBII, Sybase, and other databases can be supported in the same manner.

Job Submission

The `lpr` client program submits jobs to the `lpd` server by simply using a TCP/IP connection and sending the files to the server. The only information the client needs is the printer and hostname, and can run as a user application.

If the printer output is piped to the `lpr` client, then RFC1179 allows the output to be directly copied from the client to the server by using the `lpr -k` (for seKure) option. While LPRng supports this option, many other LPR server implementations are defective or do not support this capability. This is useful when creating large jobs, or there is are security related problems with creating a temporary file on the client host.

The LPRng clients can run as ordinary user processes; eliminates any problems with unauthorized access to files, as the client has no permission except those of the user.

However, for the `lpr` client to be compatible with *vintage* LPR spooling software (i.e.- SUN

Attribute	Match	Connect	Job Spool	Job Print	LPQ	LPRM	LPC
SERVICE	S	'X'	'R'	'P'	'Q'	'M'	'C'
USER	S		JUSR	JUSR		CUSR	CUSR
HOST	S	RH	JH	JH	RH	JH	JH
IP	IP	RIP	JIP	JIP	RIP	JIP	JIP
PORT	N	PORT	PORT		PORT	PORT	PORT
REMOTEUSER	S		CUSR		CUSR	CUSR	CUSR
REMOTEHOST	S	RH	RH	JH	RH	RH	RH
REMOTEIP	IP	RIP	RIP	JIP	RIP	RIP	RIP
PRINTER	S		PR	PR	PR	CPR	CPR
SAMEHOST			SH			SH	
SAMEUSER			SU			SU	

KEY:

CUSR	user name in connection	
JUSR	user name in control file	
RH	connecting host name	
RIP	connecting host IP	
PORT	connection origination port	
JH	host name in control file	
JIP	IP address of JH	
SA	Same Host	JIP == RIP
SU	Same user	JUSER == CUSER

Match: S = string with glob wild card, IP = IPaddress[/netmask],
N = low[-high] number range; NOT negates the test status

Figure 4: Permission Attributes

Microsystems), it must originate a connection from a *privileged* port. For this reason, when run as a SETUID ROOT program, after making a connection to the the server, the `lpr` client uses *setuid*(2) to drop the root permissions, and operates as an ordinary user program.

Several of the *vintage* `lpr` options such as the `'-s'` (use symbolic links) and `'-r'` options (remove on printing) are not supported; the symbolic link option has no effect as files are transferred directly to the server, and the remove option has caused more than one user to accidentaly delete the files that he wanted printed!

Permissions and Authorization Checking

One of the requirements of any printer spooling system is to deny access to unauthorized users and to record accounting information for authorized users. The LPRng software uses a rather elaborate permissions and authorization mechanism, similar to the ones used by computer network firewalls.

Since all spooling operations are carried out by the `lpd` server, it is the only process that needs to perform permissions checks. Permissions are checked when a connection is made to the server, and before the server performs and action or provides information requested by the various client programs. In addition, the server checks job permissions before it prints a job as well as when the job is submitted. This allows NFS based printer spooler software, which copies control and data files directly to a spool directory, to be used with the LPRng software. See the Security Considerations section for a discussion of problems related to allowing this type of activity.

Each request for service has a set of attributes and values; a list of these attributes is shown in Figure 4. Figure 5 shows a sample permissions database. Each line in the database consists of a match result and a list of attribute names and match patterns. Permissions checking is done by scanning the database in order, checking each line for a match. If all the entries on line match, then the result is the match result for the line or the current default. Note that each entry can have several alternate patterns; these patterns are tried in order until a match is found.

The `default_permission` configuration variable specifies an initial (default) permission line; additional permission databases are specified by the `printer_perms_path` configuration variable. When checking permissions for a spool queue with printcap entry, the `xu` printcap tag provides an additional set of databases to be searched. If no match is found after searching all specified databases then the last specified default permission will be used.

Attributes are treated as string, integer, or IP address values. The string patterns are based on the simple `glob` patterns of the Bourne and C shells, and use case insensitive matching with only the `*` metacharacter. For example, the pattern `A*b` will match `Ab`, and `AthisB`. IP address patterns are an address (ADDR) followed by an optional netmask (NM) which defaults to 255.255.255.255; the match succeeds if (using C language notation) `(IP^ADDR)&NM` is zero. For example, the pattern `130.191.163.0 / 255.255.255.0` matches all of the addresses in the `130.191.163.0` subnet range. Number patters are a low to (optional) high integer range.

The special pattern *char*`= pattern` matches the *char* line in the job control file against pattern. For example, `C=A*,B*,C*` will check the *C* (class) information line for a string starting with A, B, or C. The special pattern *NULL* matches missing or no information; for example the permissions entry

```
# Permissions Database
# Reject connections not in our subnet
REJECT SERVICE=X NOT IP=130.191.0.0/255.255.0.0
# Allow root on trusted hosts to have control access
ACCEPT SERVICE=C HOST=hop.sdsu.edu,skip.sdsu.edu \
    PORT=721-731 USER=root
REJECT SERVICE=C
# do not allow forwarded jobs from anybody but dickory
ALLOW SERVICE=R NOT SAMEHOST HOST=dickory.sdsu.edu
REJECT SERVICE=R NOT SAMEHOST
# Allow PC lab to spool to laserwriter
ACCEPT SERVICE=R,P,Q PRINTER=lw4 HOST=*.eng.sdsu.edu
# Let them remove jobs if from the same host
ACCEPT SERVICE=M PRINTER=lw4 HOST=*.eng.sdsu.edu SH
REJECT HOST=*eng.sdsu.edu
# if no match in other database then you fail
DEFAULT REJECT
```

Figure 5: Sample Permissions Database

`ALLOW SERVICE=R,P USER=NULL,*` allows anonymous job spooling and printing.

Spool Queues and Job Files

The main activity of the `lpd` server is centered on managing print jobs in the spool queues. A print job consists of a control file, containing user and other information, and data files containing the information to be printed. The control file format is specified by `RFC1179`; a sample job control file is shown in Figure 6. Control file names have the format `cfXnnnHOST`, where X is a letter, nnn is a 3 digit job number, and `HOST` is a host identifier. Data files names have the format `dfXnnnHOST`, where X is a letter, and nnn and `HOST` are identical to the corresponding control file.

```
Htaco.sdsu.edu
Ppapowell
J(stdin)
CA
Lpapowell
Qtl
fdfA917taco.sdsu.edu
N(stdin)
UdfA917taco.sdsu.edu
```

Figure 6: Job Control File

Control file lines starting with an upper case letter provide information and those starting with lower case letters specify a format and a data file to be printed with the format. For example, the `P` (person) and `H` (host) lines give the originating user and host name; the `I` (indent) and `L` (banner name) are used when printing the job.

The LPRng software extends the basic RFC1179 control file entries by adding `Z` (output filters options) and `Q` (original queue). The value of these options are passed to the filters that format and print the data files. For example, Figure 3a shows an example of a printcap entry (p2) with several aliases. The `lpr` command `lpr -Q -Pdouble -Zheavy_paper` will create a control file with the `Qdouble` and `Zheavy_paper` entries and sends it to the p2 printer. The output printing can use the `Q` and `Z` entries to select various paper and format options.

LPD Server Operations

The `lpd` server creates *queue server* process for each spool queue, and then waits for connections from clients. Each time a request arrives the server will create a new process to handle the requests. The `max_servers_active` configuration variable can be used to limit the number of active servers. The queue server process uses the printcap entry information and a set of control files in the spool directory to control its activities and report its actions (Figure 1). In the discussion below, *printer* is stands for the primary printer name; all files are in the spool directory unless otherwise indicated.

The Server lock file (*printer*) is used to ensure that only one server process is active at a time. The spool control file (`control.`*printer*) has the format shown in Figure 7a, and controls one or more of the spool queue related activities. Entries in this file override defaults and values in the printcap database. Note: the information shown in this file may not be present at all times.

The control file `spooling_disabled` and `printing_ disabled` entries disable spooling to the queue and printing from the queue respectively. The `redirect` entry causes the server to transfer all spool jobs to the specified remote printer. When `autohold` is enabled, the server will not process a jobs until it is released by a request from the `lpc` program.

```
printing_disabled 0
spooling_disabled 1
debug 10,remote=5,log=/tmp/log
redirect  p3@mentor
autohold  off
class     A,B
```

Figure 7a: Spool Control File

The `class` entry restricts the printable jobs to the specified class. This facility allows special forms to be mounted on a printer and only jobs which need them to be printed. The special pattern *char*`=patterns` restricts printing to jobs with a control file line starting with *char* which matches pattern. For example, `P=accounting` could be used to restrict printing to jobs from the `accounting` user.

The `debug` entry is a diagnostic and testing aid. The set of options are used used by the server to enable or disable specific testing functions. For example, `10,remote=5,log= /tmp/log` specifies a general debugging level of 10, setting the remote flag to 5, and logging to the `/tmp/log` file.

The `lpc` (line printer control) program is used to request the `lpd` server to change the spool control file values and take other actions, such as starting or stopping server processes. The `lpc` program can also request (brutal) spool server process termination, and (gentle) restarting of spooling activities.

The spool server process scans the spool queue, ordering jobs to be serviced in a first-in, first-out order within priority classes. Class A is the lowest (default) priority, and Z is the highest. When a job is selected for for servicing, the spool server forks a subserver process to carry out the actual work..

The reason for using a subserver process for per job servicing is based on experiences with a variety of UNIX implementations. Some of these implementations have *memory leaks* or *file*

descriptor leaks associated with various database and networking routines; each time a process uses these routines they open a new file descriptor or allocate some temporary storage. Unfortunately, these descriptors are never closed the descriptors or reclaim the storage. These defective functions are *firewalled* in a subserver process, which only exists while a particular job is processed. Note that the same problems exist in the `lpd` server, which also takes care to isolate these actions in a subserver process.

When a job is selected for service, the subserver process creates a job hold file to record information; job `cfA001mentor` will have hold file `hfA001mentor`. The hold file has the format shown in Figure 7b.

```
active      2743
hold        1
priority    0x873486
remove      1
redirect    p4@mentor
error       Printer timed out
```

Figure 7b: Job Hold File

The `active` entry records the process ID of the subserver process, and indicates that the job is active. A non-zero hold entry indicates that the job is being held by administrative actions; a hold value of 0 allows a job to be printed. The `lpc` *hold* and *release* commands can be used to hold and release jobs.

The `priority` field specifies an additional level of job priority; jobs with non-zero priority fields are serviced before jobs with 0 fields; the `lpc` *topq* command updates the priority value.

The `redirect` entry supplements the spool queue `redirect` information. This entry allows individual jobs to be moved to another spool queue. The `lpc` *move* command updates the redirect value.

The `remove` and `error` entries are used to solve a problem with defective or misconfigured printing software. After a job is serviced its files are removed from the spool directory. However, sometimes due to accident or intent, the files cannot be deleted, resulting in the job being endlessly printed and preventing normal operations. When a job is serviced, the job hold file is created and written in the spool directory; if the hold file cannot be modified the job is not serviced. After the job has been serviced the `remove` field is set to a non-zero value; this prevents the job from being reprinted, and the `error` field records any error conditions that might inhibit retrying servicing the job. This information is displayed by the `lpq` (line printer queue) program. After the job files have been successful removed, the server then removes the job hold file.

A *bounce* queue is used to temporarily hold jobs until they can be forwarded to a remote printer. This is useful when sending jobs to a network printer. The LPRng software `lpr` and `lpd` programs use the same algorithm to check file permissions and accessibility when sending jobs to a remote printer.

Printing Algorithm

On the surface, dealing with the printer hardware should be quite simple: the printer device is opened, the job data files are sent to the device, and the printing device is then closed. The actual algorithm used by the `lpd` server for printing a job is rather complex, in order to deal with the following problems.

1. Each printer usually has specific requirements for connection and initialization, not to mention the actual transmission of data.
2. If the connection to the printer is a serial line, *stty* (1) (or a similar function) must set the speed, format, and other characteristics. When a serial line is closed and reopened the line characteristics may be reset to some default value, requiring the line to be held open throughout the printing process.
3 The effects of the failure printing a job job should be localized to that job.
4 Different types of output such as raster plots, PostScript files, text files, etc., may require different handling when printing. This can be very device specific.
5 Multiple users may use the same printer; jobs need to be carefully separated, banner pages provided, and other administrative functions performed.
6 Administrators have a strong desire to record the printer usage so that users can be billed appropriately.

In order to handle printer specific problems, each printer has a set of filters or support programs which provide support for specific operations. For example the `of` filter will print banners, page separators, and other high level queue control functions. Files whose print format is the (lower case) character `?` will be printed using a `?f` filter; the programs corresponding to each format are found in the printcap file.

The algorithm used by LPRng is shown in Figure 8. It is similar to the original Berkeley algorithm, but not identical. Names such as `'of'` refer to entries in the printcap database and `OF` is a filter process created from the `'of'` information; `OF = filter('of') -> LP` means create the OF filter from the `of` information in the printcap file, and send it output to the `LP` filter or device.

```
LP = open( 'lp' );  // open device
OF = IF = LP;    // set defaults
if( 'of' ) OF = filter( 'of' ) -> LP;
                  // make OF filter
if( accounting at start 'as')
   do accounting;
if( leader on open 'ld' ) 'ld' -> OF;
                  // send leader
if( FF on open 'fo' ) 'fo' -> OF;
                  // send FF

// check to see if banner required
do_banner =
   (always banner 'ab'
      || (!suppress banner 'sb'
         && control file 'L' ));
if( ! header last 'hl' && do_banner ){
   BP = OF; bnr = null;
   if( banner start 'bs' ) bnr = 'bs'
   else if( banner program 'bp' ) bnr = 'bp'
   if( bnr ){
      BP = filter( bnr ) -> OF;
   }
   short banner info -> BP;
   if( BP != OF ) close( BP );
}
// suspend the OF filter
if( OF != LP ) suspend OF filter;

for each data file df in job do
   // send FF between files of job
   if( !first job && ! suppress FF 'sf' ){
      if( OF != LP ) wake up OF filter;
      'ff' -> OF;
      if( OF != LP ) suspend OF filter;
   }
   // get filter for job
   ?F = LP; // default - no filter
   format = jobformat;
   if( jobformat == 'f' or
      jobformat = 'l' ){
         format = 'f';
   }
   filter = format filter from printcap;
   if( filter ){
      ?F = filter( filter ) -> LP;
   }
   // send data file to printer
   // through filter
   data file -> ?F;
   // kill filter
   if( ?F != LP ) close( ?F )
endfor

// finish printing
if( OF != LP ) wake up OF filter;
if( header last 'hl' && do_banner ){
   if( ! no FF separator 'sf' )
      'ff' -> OF;
   BP = OF; bnr = null;
   if( banner end program 'be' ) bnr = 'be'
   else if( banner program 'bp' ) bnr = 'bp'
   if( bnr ){
      BP = filter( bnr ) -> OF;
   }
   short banner info -> BP;
   if( BP != OF ) close( BP );
}
if( ff on close 'fq' ) 'ff' -> OF;
if( trailer on close 'tr' ) tr -> OF;
if( accounting at end 'ae') do accounting;
if( OF != LP ) close( OF );
close( LP );
```

Figure 8: Printing algorithm used by LPRng

While the algorithm used by LPRng resembles the original Berkeley LPR algorithm, it has subtle differences. Before the job is printed, it is checked for the formats it uses. If there is no filter available for a data file, the job is not printed and only an error message is generated.

The printing device is opened and closed for each print job. This eliminates problems of printer failure when various network and other printers fail such that they will not work correctly until reset by a network reconnection or a device open.

The `as` and `ae` printcap entries specify a filter or format to be used to record accounting information at the beginning or end of a job respectively. For example, for a 230 byte long job spooled to printer `p1` by `john` on `pc1` the entry `as=start $P $u $H $b` will write `start p1 john pc1 230` to the accounting file. The entry `as=/usr/local/psaccnt start` will run the `psaccnt` program and wait for it to terminate. Similar action is taken at the end of a job using the `ae` printcap entry.

Each site usually has different needs for banner printing. LPRng has removed fancy bannner printing from the `lpd` server to a separate program. The `bp` (banner printer) program generates a banner for a job; users can modify the banner without modifying the LPRng software. Banners can be printed at the beginning and end of jobs.

LPRng can use *vintage* filters available for LPR and other spooling systems with a minimum of changes. The section on Filters discusses how they are accommodated.

LPRng supports multiple printers serving a single print queue. The master print queue has a `sv=server1,server2,...` (servers) printcap entry listing the server printer names; server printers have a corresponding `ss=master` (serves) printcap entry. The master spool queue server process creates a subserver process for each slave printer; the subserver processes print all jobs in the server spool queue and then terminate. As each of the subserver processes terminates, the master selects a job from the master spool queue and then creates a new subserver process. This subserver will copy the job to the server spool queue and then process the job. Note that print jobs can be directly spooled to slave spool queues, allowing users to send jobs to a server printer as well as to the master spool queue.

Filters

The LPRng software makes heavy use of filter processes for printing and other operations. A filter specification has the form

| [ROOT] [-$] *path optionsP*

Printcap printer filter entries usually drop the '|' filter indication. Normally, filters run with EUID and RUID *daemon*; the ROOT keyword runs EUID ROOT. See Security Considerations for details.

The `path` entry specifies the absolute pathname of an executable file and the `options` are a set of options to invoke the filter with. In addition to the user specified options, the LPRng software will append the configuration variable `filter_options` unless suppressed by the `-$` flag.

The options are scanned for variable substitutions indicated by `$` characters. If *key* has a non-zero length string value `X`, then `$`*key* expands to `-`*key*`X`, `$-`*key* expands to `X`, and `$0`*key* to `-`*key* `X`, i.e., a space separating the key and value. For a printer filter, if the data file format is binary `$c` expands `-c`. The substitution formats allow the user to create interfaces to *vintage* printer filters with a minimum of effort; see Figure 9 for an example. As a further aid, The printcap `bkf` (backwards filter) flag appends a list of options which are compatible with most *vintage* printer filters.

In addition to the command line options filters have the `PRINTCAP`, `CONTROL_FILE`, and `DATA_FILE` environment variables set to the printcap information, control file contents, and data file name being printed. This allows filters to use information in the control file or printcap entries with a minimum amount of effort.

By convention filters read input from `stdin`, write to `stdout`, and write errors to `stderr`. The error output is usually directed to the error logging file for the printer. Print filters have their current directory set to the printer spool directory.

Security Considerations

Security considerations were a major factor in the design of the LPRng software. Many of the problems center on the following issues.

1. Users trying to use the printer spooler software to exploit bugs in the operating system and gain root access.
2. Users trying to use the printer spooler software to gain unauthorized access to other users files,
3. Users trying to gain illegal access to printing facilities.
4. Users trying to avoid accounting procedures.
5. Denial of service attacks.

The first issue to be dealt with is the problem of ROOT permissions. All of the client LPRng programs can run as ordinary users; this eliminates a large number of attacks on system security by trying to exploit various defects in the system based on SUID root programs. The LPD server is the only program that absolutely needs to run with real UID (RUID) ROOT as it uses a *privileged* TCP/IP port to listen for incoming connections, and in most UNIX systems *bind*(2) requires EUID ROOT permissions to bind to a privileged port. (It is not recommended that a non-privileged port be used as a trojan horse user program can bind to it and impersonate the LPRng software.) According to RFC1179 a connection to a server must originate from a (privileged) port in the range 721-731.

Given this need for ROOT permissions, the LPRng code goes to extreme lengths to ensure that only the *bind*(2) calls are made with EUID root, and that all other operations are done either as daemon (server) or as user (clients). It is strongly

Filter specification:

```
path arg1 arg2 $P $w $l $x $y
  $K $L $c $i \
  $Z $C $J $R \
  $0n $0h $F $-a
```

Expanded Specification

```
path arg1 arg2  \
   -PPrinter -wpw -lpl -xpx -ypy \
   -Kcontrolfilename -LLogname -iIndent \
   -ZZoptions -CClass -JJobinfo -RRaccountname \
   -n Person -h Host -Fformat af

Note: pw, pw, etc. are from printcap entries,
  Printer, Logname, etc. are from control file lines,
  other information generated by server.
```

Figure 9: Filter Specification and Expansion

recommended that the `lpd` program not be SUID root, but should started up by the system initialization *rc* (4) scripts or a root user.

It is recommended that all client programs be run as user (non privileged) jobs. Only files accessible to the user will be read or transferred to the server. If a user wants to access a printer that requires privileged ports, it is a simple matter to create a *bounce* queue on a server that will forward a job to the remote system.

The `checkpc` (check printcap) program scans the printcap and permissions databases, spool queues, and checks permissions of files and directories. If run by ROOT with the `-f` (fix) flag set, it will try to change ownerships, create files and/or directories, and remove junk or old job files from spool queues. This program also has some portability tests built into it, and can be used to check that the target system can safely run the LPRng software.

Most efforts to circumvent accounting and permissions checks are based on forging or impersonation of another user or network host. The current version of the LPRng software depends on the various system configuration and database utilities to provide user authentication and system authentication. This is clearly inadequate, and a future release of LPRng will support encryption based authentication; the KERBEROS and the PGP systems are under active study for possible use. The method will be base on using the *filter* mechanism to invoke a set of authentication programs rather than directly incorporating the code into the LPRng software. This allows a variety of mechanisms to be used.

One of the arguments for running client programs SUID ROOT is that they are enabled to connect to the server from a privileged port, and the information provided will be authenticated in some manner by the operating system. Unfortunately, the LPRng software uses various network databases to obtain connecting host information; by attacking the the system by spoofing database (DNS) server activities, it is possible to forge authentication.

The use of NFS exported and mounted spool directories exposes the LPRng software to extreme attack. One of the assumptions made by most spooling systems is that only the *trusted* spooling software or trusted application programs will have write access to the spool directory; when the directory is NFS mounted or exported this may no longer be true. Several spooling systems operate by writing job control and data files into an NFS mounted spool directory. By appropriately forging network identification, credentials, and various RPC calls, attackers can create or modify unprotected files in the spooling directory. The ability to read information in job or other files may also give them the ability to launch other forms of attack. One of the more malicious denial of service attacks is to create a file that cannot be removed or modified; the spooler software may end up repeatedly attempting to print the file, blocking other users from using the spool queue and consuming printer resources.

In order to protect the LPRng software from NFS spoofing based attacks, the printcap `cd=`*directory* entry specifies a separate *control* file directory to be used by `lpd` for all spool queue files except the job and data files. This directory should *not* be NFS mounted or exported, and should reside on the local host file system. This directory should be carefully created so as to be accessible only by user *daemon*. Printcap and other information can be safely placed in this directory as it cannot be modified by NFS operations.

Avoiding printing accounting procedures has long been a tradition at educational institutions; while minor infringements are usually ignored, persistent and blatant offenses are worrisome. In addition, once an individual discovers a method then it apparently is rapidly copied by others, leading to widespread abuse. One difficulty faced by administrators is determining the resources used by a job. As part of the printing algorithm, the LPRng software provides a set of *hooks* to allow the invocation of accounting programs before and after the actual job is printed. For example, most PostScript printers have a *page count* register whose value can be easily read by a simple Postscript Program. By reading this before and after a job the total usage can be calculated.

However, some students have discovered that by aborting a job in the middle of its printing or by printing a job that contains information that causes the printer to hang and not report the total pages used at the end of a job they can avoid the normal accounting procedures. By recording information *before* as well as *after* a job completes such incomplete jobs can be found.

Filters are a major security loophole, as most filters are shell scripts and inherit shell script vulnerabilities. To combat this, the LPRng software defaults to running all filters either as the user or as *daemon*, and provides a predefined and limited set of environment variables. Some network printer filters need to open a privileged port and must have root permissions. This is a serious vulnerability, and the `lp=host%port` printer specification has been provided to ameliorate this problem. It has been recommended that filters run as user *nobody*, restricting capabilities to an even greater exent, and this consideration is under study.

Filters which are actually shell scripts are vulnerable to attacks using metacharacters in option strings. These are passed as options to the filter, and are then rexepanded by the shell. For example, a job spooled by the user named `'root -rf /'` (note the backquotes) would have interesting results.

The `vintage` printer filters are particularly vulnerable to attacks of this type. To combat this, the LPRng software ruthlessly purges all non-alphanumberic, whitespace and simple punctuation (minus, period, slash, and comma) characters from filter options. The raw option information is available in the PRINTCAP and CONTROL_FILE environment variables. Administrators would be wise to examine shell based printer filters for similar security loopholes.

Deliberate denial of service attacks are almost impossible to avoid. However, heavy usage of the printer system can produce almost the same symptoms. For example, when a large number of print jobs are queued it is possible to exhaust the spool queue file space. The printcap `mx` (maximum job size) entry specifies the maximum job size (in Kbytes) to be queued and the `mi` (minimum free space) entry specifies the minimum free space (in Kbytes) needed.

Summary and Acknowledgments

The LPRng software continues to evolve as users find problems and develop new printing requirements. One of the areas to be pursued is the use of encryption for end to end authentication of users and print jobs. Another is adding interfaces to other network based spooling systems. Finally, documentation and automated management continues to be pursued.

The network based interfaces for client programs almost trivialize development of user specified GUI systems. PERL scripts and Tkl/Tk based front ends can be developed rapidly and easily.

The development of the PLP and LPRng software would not have been possible without the aid and assistance of literally hundreds of users. The main developer of the software was Patrick Powell <papowell@sdsu.edu>, and Justin Mason <jmason@iona.ie> generated the PLP4.0 distribution, contributed much of the portability code, and organized the plp@iona.ie mailing list. Subscribe by sending email to plp-request@iona.ie with the word subscribe in the body. Marty Leisner <leisner@sdsp.mc.xerox.com>, Ken Lalonde <ken@cs.toronto.edu>, and Michael Joosten <joost@ori.cadlab.de> performed invaluable portability testing and debugging of the LPRng Alpha Minus release; they discovered and provided fixes for literally hundreds of bugs.

LPRng was based on PLP Release 4.0, to which the following people (in alphabetical order) contributed:

Dave Alden <alden@math.ohio-state.edu>
Julian Anderson <jules@comp.vuw.ac.nz>
Jan Barte <yann@uni-paderborn.de>
Baba Z Buehler <baba@beckman.uiuc.edu>
Lothar Butsch <but@unibw-hamburg.de>
David M Clarke <dmc900@durras.anu.edu.au>
Panos Dimakopoulos <dimakop@cti.gr>
Angus Duggan <angus@harlequin.co.uk>
Martin Forssen <maf@math.chalmers.se>
Michael Haardt <u31b3hs@POOL.Informatik.RWTH-Aachen.DE>
Eric C Hagberg <hagberg@mail.med.cornell.edu>
Paul Haldane <Paul.Haldane@edinburgh.ac.uk>
George Harrach <ghharrac@ouray.Denver.Colorado.EDU>
Stefano Ianigro <w_stef@unibw-hamburg.de>
Helmut Jarausch <jarausch@igpm.igpm.rwth-aachen.de>
Michael Joosten <joost@ori.cadlab.de>
Stuart Kemp <stuart@cs.jcu.edu.au>
Hendrik Klompmaker <Hendrik.Klompmaker@Beheer.zod.wau.nl>
Rick Martin <rickm@cs.umb.edu>
Todd C. Miller <Todd.Miller@cs.colorado.edu>
Corey Minyard <minyard@wf-rch.cirr.com>
Dorab Patel <dorab@twinsun.com>
Ed Santiago <esm@lanl.gov>
Bjarne Steinsbo <bjarne@hsr.no>
Harlan Stenn <harlan@landmark.com>
Julian Turnbull <jst@dcs.edinburgh.ac.uk>
Bertrand Wallrich <Bertrand.Wallrich@loria.fr>
Greg Wohletz <greg@cs.unlv.edu>

Author Information

Patrick Powell <papowell@sdsu.edu> is faculty in the Dept. of Computer and Electrical Engineering at San Diego State University, San Diego CA 92182, where he teaches Computer Networks, Real Time Systems, and Distributed Computing.

Justin Mason is a sysadmin for IONA, Corp. in Ireland, where he administers the various UNIX systems and fixes broken printer software.

References

Pow95. Patrick A. Powell, *LPRng – Enanced Printer Spooler Software Reference Manual,* Dept. of Electrical and Computer Engineering, San Diego State University, San Diego, CA 92182, 1995. FTP://ftp.iona.ie/pub/LPRng/, FTP://dickory.sdsu.edu/pub/LPRng/

Cam94. Ralph Campbell, "4.3BSD Line Printer Spooler Manual," 4.4 Berkeley Software Distribution, Computer Systems Research Group, U.C. Berkeley, Berkeley CA, 1994. USENIX Association and O'Reilly & Associates, Inc.

Pow95a. Patrick A. Powell, "PLP – The Public Line Printer Spooler Reference Manual," PLP 4.0 Software Distribution, 1995. FTP://ftp.iona.ie/pub/plp-4.0

McL90. Leo J. McLaughlin III, *RFC1179 Line Printer Daemon Protocol,* Internet Advisory Board, 1990.

GNU91. GNU, *GNU General Public License,* Free Software Foundation, Inc., 675 Mass. Ave. Cambridge, MA 02139, 1991.

Finding a Needle in a Virtual Haystack: Whois++ and the Whois++ Client Library

Jeff R. Allen – Harvey Mudd College

ABSTRACT

Powerful Directory Services are imperative in large networks to help keep users connected to the people and resources available on the net. This paper surveys previous work to build Internet Directory Services, and presents a set of requirements for the next generation of Directory Service technology. Next, the paper presents an overview of a new standards-track protocol named Whois++. Finally, client software written by the author is presented, and freely available server software is reviewed.

Introduction

It is no secret that the Internet is growing at an incredible pace. As a matter of fact, much of a system administrator's job is trying to keep up with this growth in all the diverse ways that it affects the organizations for which we work. With this expansion comes growing pains, as technology falls increasingly short of the demands placed on it. One technology that has fallen drastically behind in this rush of growth is Network Directory Services; the job of finding people, machines, and services on the network.

There is a loosely organized body of work meant to correct this problem, ranging from the Finger protocol (first documented in 1977) to the entire X.500 effort, dating from before 1985 to as recently as 1993. Still, we are faced with the reality that finding people on the Internet is one of those things best left to a network guru, one who knows all the right nooks and crannies into which to delve. Worse yet, the word is out to the "customers", the new class of users who are flooding onto the net, that the Internet's Directory Services aren't up to par. In *Newsweek's* Cyberscope column, the editors made the following observation: "The Internet provides myriad opportunities for procrastination. One of the best ways to avoid real work is trying to find someone's Internet address." [NW94] They go on to recommend a service named **netfind**, which works acceptably well, but falls short of the kind of ease of use required for a truly "good" solution.

With support from the IETF (as part of the WNILS working group and later, the ASID working group), another generation of researchers have attacked the problem, this time revamping the Whois system used by Internic (and previously, SRI's NIC) into a fully distributed, client/server system called Whois++. It is the author's belief that Whois++, while not perfect, will prove useful in the struggle to bring the Directory Service problem under control.

This paper introduces System Administrators to the concepts and technology of Whois++ so that they will be ready to adopt it if and when their users call for it, or when they recognize a problem in their organization that could be solved using Whois++. In particular, the paper will discuss an API and library that the author has developed to ease the task of writing innovative Whois++ clients.

The Directory Services Problem Explained

The Directory Service Problem is about connecting people to other people. There are many people on the network, and it is hard to keep all of the information about all of the people accessible to all of the rest of the people in an easy to use, easy to search directory system. Consider, on top of that, the tremendous rate of change of both the number of people, and the information about them, and the problem seems almost impossible. At all times, the problems of scaling in the system must be confronted head-on. Only through careful engineering, including application of client/server database concepts and distributed indexing technology, can a large-scale Directory Service system succeed. (Truth be told, it takes hard work, good politics, and a little luck too!)

For readers who want a more rigorous justification of the scaling problems the following may suffice: The fundamental problem is that the need for Directory Services grows as n^2 when the community increases by n members. This is due to the fact that in a group of n people, there are n^2 possible acquaintances, and if we assume the need for Directory Services among this group is roughly proportional to the number of acquaintances, then the need grows at the square of the rate of population growth. The constant of proportionality is anyone's guess, but the fact remains that the growth in the number of acquaintances is not strictly linear. Any system will have to take this characteristic of the

problem into account, most likely by providing ample scalability in the design.

In addition to concerns of raw scale, there is another reason for the demand for Directory Services. The Internet is becoming a competitive market of consumers and producers, much like the US long-distance carrier market did in the last decade. To a professional who uses the network for business, the change in address necessitated by a switch to a lower-cost provider might mean lost contacts. Directory Services aren't just an interesting research challenge anymore. Users are in need of a quick solution to their problem: they want every contact to be able to find them quickly and reliably, no matter where they "reside" on the global Internet.

Due to the growing administrative complexity of managing names and addresses in the rapidly expanding Internet, a decentralized system of naming authorities has been created. These groups (like Internic and RIPE) all possess useful information about network entities, but it is hard for users to access, since it is spread across the network. This is a case where a strong distributed network Directory Service would serve users well. In fact, a protocol called RWhois is being developed to meet the immediate need of scaling the existing Whois system up to handle multiple naming authorities. RWhois, however, is tightly wed to the current Whois system. [RFC1714]

Finally, coming up with a scalable, extensible directory system may give other researchers the tools they need to solve other resource discovery problems. Effort is already being put forward by the folks at Bunyip Information Systems to merge some of the capabilities of Archie with the distributed characteristics of Whois++. Those working on a key distribution facility for a public key cryptographic system may also want to look into Whois++. As with any well designed tool, the eventual uses cannot even be imagined by the present day users.

Existing Systems

There are a number of systems currently in place to assist in the demand for Directory Services, but none of them have the scalability and ease of use required to solve the problem in both the short and long term.

Perhaps the earliest attempt of all at solving the Directory Service problem in the Internet was the Finger protocol, defined in RFC 742 in late 1977. This simple protocol was easy to design, easy to implement, and most importantly, solved the problem at hand nicely: it allowed the researchers on a handful of machines to find out who was logged into a handful of other machines on the net. From there, it evolved into a quick and easy way for people to distribute information about themselves to others. It remains one of the primary ways PGP keys are exchanged. With regard to solving what we now understand as the very complicated problem of Directory Services, Finger is a complete failure. In its time, it was a nice little application of the evolving network.

Why doesn't Finger fit the bill for a network-wide Directory Service? The biggest problem is that there is no cross-indexing in the system of servers. There are literally millions of servers out there, each holding a little bit of useful information. The problem is getting the right server, and retrieving the information of interest. Because the results of a Finger query can't be reliably parsed by a computer program, the arduous task of searching the global Finger database can't even be automated. It has to be done by hand by an experienced network user, one who knows how to find the information they are after.

Like Finger, Whois was a protocol designed to fill an immediate, pressing need. The Network Information Center (NIC) at SRI was building a database of useful information about Internet users. To share this information, a stateless, one-shot TCP based protocol was defined. It works just like Finger, except that a more advanced syntax for searching was established. To this day, the Whois servers at **nic.ddn.mil** and **rs.internic.net** get thousands of queries a day. (The InterNIC alone estimates they received 70,000 hits a day during the month of June, 1995.) [INIC]

There were several problems with Whois. First, the protocol was never really documented as an official Internet Standard. Instead, RFC 954 reads like an instruction manual for using the server. Since no reference implementation of the server was ever released for network-wide use, several incompatible versions of servers that implemented Whois-like services sprung up. That there was no reference server is understandable, since the NIC database is stored in a commercial database, and any code released to the public would be essentially useless without the same commercial server and database configuration. Even if a compatible group of servers had been installed as a result of the growth of Whois' popularity, there was still no cross-indexing in the system, hobbling its effectiveness for large-scale searches.

One feature that Whois introduced to the Directory Services field was the concept of handles. In database terminology these are "primary keys" for the server's database. Handles are alphanumeric identifiers that are unique within a given Whois server. They provide an easy way to come back to data retrieved earlier. In some cases, the rules used to make handles are so predictable that a search can be formulated in the form of a handle lookup, yielding a quick, focused search. This is an important feature that can be seen in all contemporary Directory Service schemes.

The ISO/OSI solution to the Directory Services problem came in the form of X.500. The X.500 development effort was spawned from the work on X.400, the OSI Messaging standard. It became clear to the X.400 developers that to make a user friendly mail system, a strong directory service would be required. This fact remains true today: behind most successful LAN e-mail systems lies a proprietary Directory Service system of some type. Typically, they are small and hard to manage, which makes them unsuitable as candidates for an Internet Directory Service. In e-mail systems without an integral Directory Service, like UNIX mail, one of the biggest problems users face is finding the right address to put on their e-mail.

When the X.500 effort got up to speed, people finally realized what a hard problem wide-scale Directory Services is and threw the heavy artillery at it. Many man-months of work went into writing the first X.500 specification. The result was a system that seemed to cover all the bases, dotting all the i's and crossing all the t's. The cost was complexity: the X.500 specification is hard to understand in its entirety and even harder to implement completely and correctly. Writing clients for the system requires understanding several layers of the OSI protocol, and mastering the TCP/IP interface used to bridge the gap between the Internet and OSI worlds.

Many organizations around the world use X.500 and/or systems derived from it to handle their Directory Service needs. It is by no means dead, and should certainly not be discounted. With that said, X.500 has had a very low acceptance within the networking community. In the opinion of the author, this failure to gain market share is due to three factors: complexity, politics, and search performance.

Due in part to the complexity of the protocol, there have been few servers made available in the public domain that support X.500. Quipu, the X.500 server that was distributed with ISODE, was poorly supported and hard to use as a result of its status as a research project; there were simply no resources to make it user friendly. Some commercial enterprises have invested in producing X.500 systems, but even so, there has been little growth in the use of X.500. One large company that has publicly endorsed the standard is Novell. However, the NetWare Directory Services system, which is based on X.500, operates over non-standard transport layers, and is not being deployed into a global infrastructure. Thus, even a high-profile player like Novell has not been able to make an impact in the use of X.500 on the public data networks.

Sadly, in the international standards process, sometimes political problems overshadow the technical ones. X.500 was unfortunately caught up in the heated Internet/OSI wars of the late 1980's, and had a slow start out of the gate as a result. That immense amounts of effort were lost is regrettable, but we must push forward, learn from the past, and try to launch a new Directory Service under more favorable political circumstances.

By far, the biggest failing of X.500 is its inability to deal effectively with large searches and multiple directory organizations. This is the kind of problem that could only be discovered through the limited real-world use X.500 has seen in the last few years. Because there is no shared information between servers, queries must be flooded out to all servers in the tree in an very inefficient manner. This causes unreasonable delays and large network cost for even fairly simple requests. Thus, at the highest levels of the tree, possibly where it was needed most, searching had to be curtailed or even turned off. [WEI95]

One other surprising development has come on the Directory Services scene in the last two years. The World Wide Web seems to be capable of some of the same features that we might demand in a new Directory Service. Data (in the form of user home pages) is stored all around the net in a highly distributed fashion. It is cross-indexed in many ways by many different servers, including Lycos, Yahoo, and Open Market's Commercial Sites Index. [LYCOS, YAHOO, OM] Perhaps the best thing about using the Web as a user directory is that the users are in total control of the data. This means that data is more likely to be quickly updated to reflect changing circumstances. The down side, though, is that the data will likely be completely unintelligible to intelligent clients, thus making some the of the very interesting features of a directory service system inaccessible. For instance, it won't be possible to make a user interface in which you double-click on a user's e-mail address to begin writing a letter to them. The browser simply won't know which bytes are the e-mail address, and which bytes are the user's favorite quote by Frank Zappa.

The ideal reconciliation of the two systems (Web based dissemination of user information, and structured, searchable Directory Services) will be to make one of the attributes stored by the Directory Service a URL pointing to the Web-based information about the user. This way, a structured search can be made for users, and once the desired person's record is found, one click of the mouse might take you to their homepage. A different click of the mouse might address a waiting e-mail message.

A Perfect Directory Service

Judging by the shortcomings of the existing systems, there are four characteristics that the next-generation system must have:

1. It must organize data into collections of attribute/value pairs, so that machines can parse the information automatically.
2. A new system must be distributed at all levels. Data storage and indexing need not be

separated, but they both must be distributed across the network to withstand heavy loading, and to provide uninterrupted service.
3. The new system must support fast and efficient searching at all levels. Without large-scale distributed indexing, this goal will be unattainable in a huge network like the Internet.
4. The organization of the data and indices in the system must be able to change over time as demands change. Indexes that cater to special interests should be possible.

The designers of Whois++ obviously had a set of goals like this in mind, since Whois++ fulfills each one nicely. This should not be surprising, of course; Whois++ is a next-generation directory service, meant to incorporate the lessons learned from the previous body of work.

The Whois++ System

As hinted above, there are really two distinct problems designers face when trying to create a Directory Service. First, they must deal with the raw data, defining protocols to transmit it while retaining automatically-parseable attribute/value pairs. Second, they must develop indexing and searching protocols to allow users to quickly find the data of interest. The designers of Whois++ divided the design into two conceptual pieces, one to serve data, and one to index it. In reality, these pieces can be implemented in the same server, so that a given server can serve local data and index the data of remote servers too.

The design of the database server is relatively straightforward. It is the part of Whois++ most reminiscent of the original Whois service. The main idea is that the database server will return templates, which are collections of attribute/value pairs identified by a template handle. Each server in the system has a server handle, which will eventually be assigned by the Internet Assigned Numbers Authority (IANA).[1] These handles are unique among all servers in use on the Internet. Within an individual server, each template handle must be unique. This makes it possible to uniquely identify any template, anywhere on the Internet, using just a server handle and a template handle.

Within a template, attributes are distinguished by attribute names. Since they are transmitted in full ASCII text, and are often stored in the server the same way, they are arbitrarily extensible. The server administrator can add attribute names to the server's templates as they are required. This begs the question of who controls the definitions of attribute names, and how do they impose their will on the various server administrators? The solution offered by Whois++ is typical of the Internet community: there is no schema administration authority. Various IETF working groups will likely publish advisory RFC's to help new administrators choose reasonable attribute names. It is the author's opinion that ultimately, the Whois++ client developers will have control over the schema. After all, what use is a fancy new attribute name if no Whois++ clients will recognize it and display it usefully?

From the point of view of the client, retrieving all that data is all fine and dandy, but the important thing is to be able to succinctly search for records in the database. The searching syntax is based on the

[1] Currently server handles are being registered by Patrik Faltstrom, <paf@bunyip.com>. Discussions are underway with IANA to find a way to assign Whois++ server handles without significantly impacting their existing workload.

```
C: <connect to muddcs.cs.hmc.edu, port 5050>
S: % 220-This is muddcs running Bunyip-Whois++: DIGGER 1.0.2
S: % 220 Ready to go!
C: handle=jeff
S: % 200 Search is executing
S: # FULL USER CSHMCEDU0 JEFF
S:  NAME: Jeff R. Allen
S:  EMAIL: jeff@hmc.edu
S:  ORGANIZATION-NAME: Harvey Mudd College
S:  DESCRIPTION-URI: http://www.cs.hmc.edu/~jallen
S: # END
S:
S: % 226 Transaction complete
S: % 203 Bye, bye
S: <disconnect>
```

Figure 1: This is a transcript of the retrieval of a single template from a Whois++ server. The server handle for this particular server is "CSHMCEDU0". Lines preceded with "S" come from the server. Lines preceded by "C" come from the client.

original Whois protocol. The syntax is specified in exacting detail in the protocol specification, so it would be pointless to cover it completely again here. [DEU95] Basically, a search string is composed of tokens from the template(s) that you'd like to match. The keywords **and**, **or**, and **not** can be used to modify the search. To further constrain where in the template a token can match, attribute identifiers can be used. Thus, a search for "**Name=Smith**" will not match a record in which the only "**Smith**" token is in the "**Postal-Address**" attribute. Finally, a specific template can be retrieved by using a handle search, assuming the user knows the handle. The form for this type of search is predictable: "**Handle=** *handlename*". Unless specifically requested to be case-sensitive, all matches are case-insensitive. Attribute matching is always done case-insensitively.

When a search is too broad, it may return many more hits than are actually useful to the user. In some cases, searches can be devised to return virtually every record stored by the server. To prevent simple overloading by broad searches, and malicious attempts to download the entire database, Whois++ servers enforce several constraints on the searches. The most important is "Max-Hits". An absolute limit is set on Max-Hits by the server administrator. No client can ever receive more than this number of templates in response to a single request.

Without additional cross-indexing technology, however, Whois++ is not much better than Whois, or Finger for that matter. The cross-indexing capabilities of the protocol are what make it so special, and may in the long term, allow Whois++-based systems to solve problems not directly related to Directory Service. The cross-indexing takes the form of centroid passing. In physics, the centroid of an object is the center of all mass, a kind of balancing point. In the Whois++ world, it's a list of tokens that represents all of the words known by a server. More precisely, the centroid of a particular template type in a server is the collection of all tokens occurring within all templates of that type. The example in Figure 2 may make the definition clearer.

If the server contains these templates:	Record 1 Template: Person First-Name: John Last-Name: Smith	Record 2 Template: Person First-Name: Joe Last-Name: Smith	Record 3 Template: Person First-Name: John Last-Name: Jones
Then the centroid will look like this:	Template: First-Name: Last-Name:	Person Joe, John Smith, Jones	

Figure 2: The centroid for the three records shows that all the tokens originally present are accounted for, even though the centroid is much smaller.

```
C: <connect to muddcs.cs.hmc.edu, port 5050>
S: % 220-This is muddcs running Bunyip-Whois++: DIGGER 1.0.2
S: % 220 Ready to go!
C: smith
S: % 200 Search is executing
S: # FULL USER CSHMCEDU0 SMITH
S:  NAME: Robert Smith
S:  EMAIL: Robert_Smith@hmc.edu
S:  ORGANIZATION-NAME: Harvey Mudd College
S: # END
S: # SERVER-TO-ASK CSHMCEDU0
S:  Server-Handle: CSHMCEDU5
S:  Host-Name: MUDDCS.CS.HMC.EDU
S:  Host-Port: 5055
S: # END
S: % 226 Transaction complete
S: % 203 Bye, bye
S: <disconnect>
```

Figure 3: This transaction shows a Whois++ client/server interaction in which both a template and a referral are returned. It is the client's responsibility to carry out the additional query suggested by the server named "CSHMCEDU0" on the server named "CSHMCEDU5".

A centroid represents the set of knowledge the server has about a its domain of the distributed database. Let's call this little server with the three templates above "Server A". If its centroid were passed to another server (call the receiver "Server B") responsible for indexing all the Whois++ servers on the network, it would be immediately obvious to Server B that Server A can't help with a query like, "Last-Name=Schwartz". This is because "Schwartz" doesn't appear under the "Last-Name" attribute in the centroid that Server B received from Server A.

As soon as servers start passing centroids, a kind of order, or hierarchy, develops. Those servers with more substantial centroids (gathered from several subservient servers) are more likely to be able to match a query. However, when they match a query based on data from a remote server, there is no way for them to reconstruct the template to be able to present it to the client. Nor does the master server even have the authority to do so; for all it knows, the template may have changed in the subservient database since the centroid was received. Instead of attempting to return all templates that match a query (an impossible feat, given the information available in a centroid), servers are allowed to return referrals to other network servers that may be able to fulfill the query. Figure 3 shows a referral from a master server for Harvey Mudd to a subservient server, also at Mudd.

Directory Services in the past (notably X.500) have suffered because their indexing structures were fixed by the design. The Whois++ design attempts to get around the problem by easing the restrictions on server-to-server connections. Because servers can pass centroids in virtually any configuration, multiple indexing-server configurations are possible. Since the client is responsible for tracking cross-references within the global database, it can detect loops in the references it receives. Thus, there is no need to protect the system's hierarchy from loops. Instead of constructing a strict server tree, administrators will create a server mesh.

Currently, those servers which are running are configured as a strict tree, with the server at **services.bunyip.com**, port 63 as root. However, it isn't hard to see how a parallel mesh might be useful, one that only indexes commercial entities, for instance, or one which will specialize in templates which represent files available for anonymous FTP.

We have reviewed both the database server and the index server. The only piece of the system left to explore is the client. Whois++ clients will likely come in all shapes and sizes, as opposed to the very limited clients available for Whois today. They will also likely be hidden deep in other applications, which will benefit from using the protocol. Whois++ clients will be able to make use of the data returned from a Whois++ server in ways Whois clients were never able to. For instance, an e-mail application might have a built-in Whois++ client. At the "To" prompt, the user will request help finding a user's name. By making a Whois++ query, they will find the name they are looking for. The client software will be able to scan the attribute/value pairs that are returned and find the one for "E-mail address". With a double-click (or a drag-and-drop, or whatever) the user can add the recipient to the message. This type of feature is something Microsoft Mail and Lotus cc:Mail users have had all along, but they have never had the entire Internet indexed via an Internet standard protocol.

What's involved in writing a client? A client needs network control code, to make and break connections to servers. It needs to parse the slightly more complicated messages Whois++ servers return. Finally, it needs to manage the search, so that server loops are avoided, and so that searches get expanded in a sensible way to make sure the requested information is found somewhere in the mesh.

Because so much of the intelligence required to conduct a distributed query has been designed out of the server and into the client, it will be somewhat harder to write Whois++ clients than it was to write clients for previous services. However, with a general purpose, easy to use API (and its implementation, a Client Library), writing clients could become easy. The details of Whois++ server interaction, loop detection, and query management can be left to the library, while the programmer concentrates on a good user interface, or on the useful application of the retrieved information. Whois++ clients may not always have user interfaces, either. Any program that uses the Whois++ protocol may profit from use of the library. For instance, an X.500 to Whois++ gateway daemon might make use of the library.

The Whois++ Client Library (WCL)

The Whois++ Client API specifies a set of data types and function calls used to interact with Whois++ servers. The implementation (written in C with an interface to Perl 5) makes it easy to write Whois++ clients. The library has the following features:

- A server cache, to amortize high TCP startup time across multiple queries.
- Easy-to-use exception handling using callbacks.
- A full implementation of the Whois++ mesh traversal algorithm. [FAL95]
- No hard-coded limits on template, attribute, or value size.

The Whois++ Client Library (WCL) comes with a text-based client for use for both testing the library, and as a ready-to-use Whois++ client. It also functions in one-shot command-line mode for use in Perl 4 and shell scripts. A prototype HTTP-to-Whois++ gateway geared to making user lookups possible via a friendly Web-browser interface is also included

with the distribution, demonstrating just how easy it is to put a nice user interface on the pre-existing library code.

The library compiles on SunOS 4.1.3, Solaris 2.x, and Irix 5.x. It is POSIX- and ANSI-compliant source, which should integrate easily with most development environments. See the section named "Software Availability" below for more information about how to get the package.

The Library in Action

To whet the reader's appetite for the library, two example uses of WCL will be described here. The first is the Whois++ to HTTP gateway mentioned above. This type of application is certain to make users happy, but how can an overworked system administrator benefit from Whois++ technology? The second example shows a subroutine that could be added to a Perl 5 user creation script to derive the new user's vital statistics, given a handle in the existing Whois++ server.

The first thing a user of the HTTP to Whois++ gateway sees is a forms-based representation of a Whois++ query. Upon submitting the form, the Whois++ query takes place. If there are several matches, an intermediate page requesting a selection pops up. Once the user has narrowed the query to a single template, the system returns and displays a page describing the user. If the user has made the required information available, a hypertext link to their homepage, a picture, and a "mailto:" link are all included by the gateway.

For the hard-core sysadmin, who prefers not to use a GUI, here's a more useful tool: a Perl 5 subroutine which can automate the information-gathering part of a user creation script. This example assumes that a corporate database representing all users is already available via a Whois++ server. (See the section on "Whois++ Servers" below for an idea on how this might be accomplished.) Furthermore, it assumes that the handles in the database are employee identification numbers. System Administrators who work for academic institutions may want to think of these assumptions in terms of a "student database" and "student numbers". Figure 4 shows a rough sketch of what the relevant parts of the script might look like. The first line imports code needed to make sure that the functions will be autoloaded at the appropriate time. After that, a prototypical subroutine call is shown. This call would likely come near the beginning of the script, after the employee number has been read from user input, or from a file. Finally, the subroutine is shown. In this case, the library is only being called upon to offer server management and template parsing services. Since we know that the employee number must map to a template on the local server if it is valid, there is no need to go off probing other servers in the mesh. Finally, the name and uid are returned. In the case of an invalid employee number, a list consisting of a pair of empty strings will be returned. The calling program can then take appropriate action.

Whois++ Servers

Like all protocols in the Internet suite, a reference server of one sort or another has existed throughout the development of the protocol to help make sure that the grand ideas were actually implementable. The reference server (currently the only one available, though that's likely to change in the coming months) was written by Patrik Faltstrom of

```
use WCL;
[... user code goes here ...]
($uid, $name) = getUserInfo($employee_num);
[... rest of script goes here ...]

sub getUserInfo {
  my ($emp) = @_;
  my $slot, @res, %av;

  $sid = wclMakeServid($serverhost, $serverport) if (! defined($sid));
  $slot = wclGetServer($sid);
  @res = wclParse(wclCommand($slot, "handle=$emp"));

  # 4th element of result is a list of the a/v pairs. This conversion
  # implicitly loses ordering and repeated keys, which are defined by the
  # protocol to be significant. We choose to ignore them for this example.
  %av = $res[3];
  return ($av{"UID"}, $av{"Name"});
};
```

Figure 4: A subroutine to retrieve user account information from an existing corporate database for use in an account creation script.

Bunyip Information Services, Inc. Bunyip has generously allowed the Internet community free use of the server (named 'Digger'), although its code is copyrighted code and may not be redistributed. Availability of the most recent version is discussed in the section named "Software Availability", below.

Digger is written in C and uses an SQL database for the back end, where the actual data and centroids are stored. Digger currently supports two database backends (Oracle and mSQL), and can be ported relatively easily to other database systems. mSQL is a relatively new publically redistributable shareware SQL server written by David J. Hughes of Bond University.[2] Digger ships with the newest version of mSQL, though it is also available separately from Bond University. [BUNYIP, MSQL]

Installing Digger is a breeze, thanks to Patrik's use of GNU autoconf and a clever INSTALL script. Installing mSQL is just as easy. Because all of the code is written to the POSIX specification, it should be easy to get the servers running on other fairly modern machines. As usual, both authors welcome e-mail describing any minor difficulties encountered during the installation process. Both packages are known to build correctly on SunOS 4.1.x, Solaris 2.x, HP-UX, Linux and OSF/1.

The biggest job a Whois++ server administrator faces is acquiring and formatting data into a form suitable for Digger's template insertion program. There are both technical and political problems here that sysadins will need to solve locally. Bunyip has promised a set of scripts to aid template conversion (send e-mail to digger-info@bunyip.com for more information about these scripts). Most administrators will want to set up a system which provides periodic updates from a central database, since it is important to keep the Whois++ server's data up-to-date with respect to the main database.

A truly adventurous SQL hacker may like to link Digger directly to an existing user database using the internal interfaces to SQL that Digger provides. Though there are no known examples of this type of project underway, only small pieces of code should theoretically be required, and the benefit (no time-delay induced inaccuracy between the databases) should outweigh the investment. Perhaps a future LISA paper will describe a project like this.

Administrative and Legal Challenges

Because of the perceived potential for abuse, developers of electronic Directories have often faced opposition from non-technical, but nonetheless interested parties, thus compounding the difficulty of the Directory Services Problem. This is a part of the problem that needs to be faced, perhaps even more urgently than the technical aspects.

Various groups, including privacy-advocacy lawyers and law makers, corporate executives, and academic administrations have all gotten involved in the fray at one time or another, each pushing essentially the same argument: electronic directories are an infringement on a person's right to privacy, and must therefore be unconditionally blocked.

The fundamental problem with the argument that electronic directories invade people's right to privacy is that less visible, but highly invasive directories already exist at the disposal of the privileged few. The entire Direct Marketing industry revolves around managing, trading, processing, and building lists of names. These directories are not made available to the public for useful ends. Instead, literally millions of tons of unsolicited mail is sent to the "lucky" members of these mailing lists. Other examples of privacy invasions from electronic directories surround us: credit reporting agencies, credit-card records used to create consumer spending histories, etc. Finally, it's helpful to ask oneself, "what's the difference between the Directory Service offered by a telephone company and one offered over a computer network?" If the telephone company can provide such a useful service, network providers certainly should be able to.

It all comes down to the rights of the person whose privacy will allegedly be invaded. All of the potential good that a public access electronic Directory can do, in this author's opinion, is worth the risk that a little bit of electronic privacy might be lost. But that's just the point: this tradeoff is an individual decision, and every person who is listed in every directory has a right to control how much information is available, and to whom. In addition, every user always has the right to submit changes to the data, and have their records promptly updated. These rights are respected by the other maintainers of lists. The Direct Marketing Association provides a registry of people who want to be excluded from the industry's "service".[3] Telephone companies provide unlisted numbers as a regular service to subscribers. Electronic Directory providers can and should provide the same types of service.

These principles of user control and notification are discussed in RFC 1355 and RFC 1295. [RFC1355, RFC1295] Administrators would be well

[2] mSQL is a very important, very useful piece of software which the Internet has needed for some time. David Hughes has earned the right to license his software for a small fee. Please read the license which comes with his software carefully and comply with it, if you choose to use his software.

[3] Call the Mail Preference Service at (212) 768-7277 and ask to be added to their Suppression File. For more information about protecting your privacy, see the FAQ on junk mail posted occasionally by Chris Hibbert <hibbert@netcom.com> to misc.consumers.

advised to review and share these particular documents with management personnel before attempting to put a large database online for public use. Taking the time to put together a coherent policy on how users relate to the data being published about them may reduce future problems.

It's Not Just For People Anymore...

With a service as general and powerful as Whois++, data of virtually any type can be indexed and served via the Whois++ protocol. Just some of the possible applications include the items below.

Public-key encryption system key-servers

While it will likely be impossible to securely offer full key escrowing services via a Whois++ server, the protocol will be useful to handle insecure key exchanges like those used by PGP. In this case, the Whois++ client/server system would simply automate what is already commonplace on the Internet today. Key exchanges are usually manually carried out via the Finger or HTTP protocols.

URN-to-URL translators

Some plans to supplant Uniform Resource Locators (URLs) with more general Uniform Resource Names (URNs) call for an infrastructure of servers to translate the various tags. These servers must come up with the closest, fastest, or cheapest URL for a given URN, and may be called upon to provide reverse mappings too. Whois++ should be up to the challenge, even though the translation database will be widely distributed and quickly changing.

Interpedia search engines and/or SOAP managers

Plans for an Internet-wide, publically authored encyclopedia representing the knowledge of all the Internet's users call for careful indexing and very fine-grain data distribution. In addition, the management of SOAPs, or Seals of Approval may be a problem suited to Whois++ technology. [RHINE]

Conclusion

It is the author's opinion that the Whois++ architecture will be a useful step forward in Directory Service technologies, enough so that it is worthwhile to develop clients for it to spur the market for Whois++ servers. With the use of the Whois++ Client Library, it should be easy to produce powerful, interesting Whois++ clients to solve the problems of a growing Internet. And with powerful, easy to use clients, the market for servers (both free and commercial) will develop.

Whois++ and WCL will be up to each of the challenges above. All that's required is a little imagination and some hard work. WCL eases the burden, leaving the programmer free to work on the challenging problem of managing and presenting the data in a useful way.

Software Availability

All of the relevant URLs are provided in Figure 5. The version numbers are correct at the time of this printing, but they will change over time. With the exception of mSQL, all of these products are free, copyrighted works. There is a small shareware fee for mSQL.

Author Information

Jeff R. Allen is a full-time student in his final semester at Harvey Mudd College in Claremont, CA. His computer-related interests include stupid Perl tricks, innovative user support, and single-handedly solving the Directory Service Problem, though graduating must take priority to all others at this time. When he is away from computers, Jeff likes to read, unicycle, and plan pranks against CalTech (though they seldom actually see fruition). E-mail messages, including job offers, are gladly accepted at: jeff@hmc.edu.

References

[ALLEN95] Allen, Jeff R. "Whois++ Client API v2.0a" (work in progress) *http://www.cs.hmc.edu/~jallen/wppcl*

[BUNYIP] Bunyip Information Services, Inc. Digger home page. *http://services.bunyip.com:8000/products/digger/digger-main.html*

[DEU95] Deutsch, Peter, Rickard Schoultz, Patrik Faltstrom, Chris Weider. "Architecture of the WHOIS++ Service" (work in progress) *ftp://ftp.internic.net/internet-drafts/draft-ietf-asid-whois-arch-03.txt*

[FAL95] Faltstrom, P., R. Schoultz, C. Weider. "How to interact with a Whois++ mesh". (work-in-progress) *ftp://ftp.internic.net/internet-drafts/draft-ietf-asid-whois-mesh-01.txt*

[INIC] Personal correspondence with Internic engineers at: <action@internic.net>, July 1995.

API:	http://www.cs.hmc.edu/~jallen/wppcl
	ftp://ftp.hmc.edu/pub/research/wppcl/api.ps.Z
WCL:	ftp://ftp.hmc.edu/pub/research/wppcl/wcl-2.0a.tar.Z
Digger:	ftp://ftp.bunyip.com/pub/digger/software/digger-1.0.4.tar.gz
mSQL:	ftp://ftp.bond.edu.au/pub/Minerva/msql/msql-1.0.7.tar.gz

Figure 5: Where to find the software

[LYCOS] Mauldin, Michael L. "Lycos, The Catalog of the Internet." *http://lycos.cs.cmu.edu*

[MSQL] Hughes, David J. mSQL Information and Distribution. *ftp://ftp.bond.edu.au/pub/Minerva/msql*

[NW94] *Newsweek*, December 5, 1994, page 10.

[OM] Open Market Inc. "Open Market's Commercial Sites Index" *http://www.directory.net*

[RFC742] Harrenstien, K. RFC 742, NAME/FINGER. December 30, 1977. *ftp://ftp.internic.net/rfc/rfc742.txt*

[RFC954] NICNAME/WHOIS. K. Harrenstien, M. K. Stahl, E. J. Feinler. October 1, 1985 *ftp://ftp.internic.net/rfc/rfc954.txt*

[RFC1295] The North American Directory Forum. User Bill of Rights. January 1992. *ftp://ftp.internic.net/rfc/rfc1295.txt*

[RFC1355] Curran, J., Marine, A. Privacy and Accuracy Issues in Network Information Center Databases, August 1992. *ftp://ftp.internic.net/rfc/rfc1355.txt*

[RFC1714] Williamson, S., Kosters, M. Referral Whois Protocol (RWhois). November 1994. *ftp://ftp.internic.net/rfc/rfc1714.txt*

[RFC1758] NADF Standing Documents: A Brief Overview. The North American Directory Forum. February 1995. *ftp://ftp.internic.net/rfc/rfc1758.txt*

[RHINE] Rhine, Jared. Interpedia Research information. *http://www.math.hmc.edu:8088/interpedia*

[WEI95] Weider, Chris, Jim Fullton, Simon Spero. "Architecture of the Whois++ Index Service". (work in progress) *ftp://ftp.internic.net/internet-drafts/draft-ietf-wnils-whois-05.txt*

[YAHOO] Filo David, Jerry Yang. "Yahoo." *http://www.yahoo.com* EM

Capital Markets Trading Floors, Current Practice

Sam Lipson

ABSTRACT

Financial trading has been described as "technological warfare". Whether or not you believe the metaphor, trading systems certainly involve a tremendous need for highly reliable, interruption free, real-time data.

Real-time, in this case, means presented accurately and without delay. Current practice of medium to large trading floors often involves Unix based workstations and significant attention to reliability. There are issues to be addressed during the construction of the physical facility, the planning and installation of the systems, and various rules of the road for support personnel. This paper attempts to address many of these areas.

In this paper some of today's common practices and the rationale behind them will be described. It is the goal to take some of the mystery of out the black art of trading floor design and support.

The ideas in this paper have been installed on several of the largest trading floors in Boston, MA (USA) covering the market segments of Foreign Exchange (F/X), derivatives, equities, fixed income, and (US) government securities. The only market segment not represented is commodities.

Introduction

In order to understand some of the constraints placed upon trading systems it is useful to understand a small amount of how capital markets work (capital as in money, or currency). In each transaction there is a buyer and a seller, like any auction system the seller(s) present their merchandise asking for a certain price, and the buyer(s) bid the amount they are willing to pay.

Eventually, a price asked for (by a seller), and the bid (from a buyer) meet, and a transaction is consummated.

A market consists of a number of buyers and sellers, all of whose primary motivation is to make a profit.

The real-time requirements come from several distinct areas. One is the need to give correct prices to customers – quote a wrong price, and you may lose a customer or run into regulatory difficulties; another is to drive analytics or proprietary trading applications. There are also some styles of trading called momentum investing, and arbitrage – which rely on nearly instantaneous execution of trading decisions based on market motion or short-lived inefficiencies. Getting your trade in first is often the difference between making a profit and being out of the game entirely.

Today's capital markets operate on a tremendous scale. The foreign exchange market alone (foreign currency buying and selling) is approximately 1 Trillion dollars per day. F/X is often speculative, as only about 10-15% of the F/X transactions per day are actually required for commerce (i.e., import/export).

We build trading systems, which normally involve Unix workstations on the desk (in our case Sun SPARC, though other platforms are used in this application), real-time data feeds, and significant attention to reliability throughout the design.

Proper reliability design in a commercial environment (and specifically financial services) – means that you have to understand the cost of a failure. Once you understand the cost of a failure, and the frequency or likelihood of that failure, then you can make a business case for spending money (or effort, which later equates to money) in order to mitigate the effects of such a failure. The trading community makes their living by taking and controlling risk, and hence can often provide more than just budgetary guidance as to whether additional expenditures are justified.

In many parts of this paper we refer to the cost: either budgetary or business impact. What we are building here is a business infrastructure. Ideally, the infrastructure for a profitable business. As in any business, our profits (trades) need to amortize the cost of what we do. Since we are building infrastructure for very sophisticated financial people, we constantly ask for expert advice on matters of expenditure, risk/reward, and business impact.

You will see examples of where we apply systems, software or hardware redundancy in ways which may seem somewhat archaic, especially in an era where systems with internal redundancy are

available. Once again, there is a cost/benefits trade-off. By spending additional money on the types of external redundancy we describe, it is often possible to get many of the benefits traditionally attributed to open systems. While there are internally redundant open systems on the market today (e.g., Stratus FTX), their use is not as widespread as the systems discussed here.

It is hard to pinpoint a specific reason for this, but it is a fact of life that it is often easier to grow a small system into a big one, than it is to build a big one from scratch. Financial people tend to like to make a small "trial" investment, and then grow it if it proves fruitful. This is yet another vote for "scalable" technology. Also, our Wall St. colleagues often have little patience for technology risk, and instead like to save their "risk exposure" for the financial domain.

A fair amount of the technology presented here would not be considered on the leading edge. Nonetheless, it has been chosen for it's proven utility, economics, and sometimes the sheer availability of skilled people for it's care and feeding. While many parts of the Wall St. community thrive on being at or beyond the state of the art, the bulk of the trading community focuses more on building highly reliable systems with a minimum of built-in obsolescence.

There are a multitude of possible decisions. Many are focused by the size of the installation, or the budget available, and sometimes there are corporate standards, or management guidelines. Hopefully, in each case the principles that guide your hand have sound business or technical factors behind them.

Divergent Routing of External Connections

Our trading systems revolve around a number of data feeds. These are commonly leased lines (though other techniques exist, including satellite and radio receivers), through which various data vendors and exchanges transmit market data. A simple example is presented by the equities market where the bid and ask prices for a stock, such as IBM, from the New York Stock Exchange (NYSE) are transmitted in real-time via a data feed.

Because it is absolutely essential to provide this data with near 100% reliability, the main feed link is always backed up by a secondary. We divergently route the primary and secondary links (i.e., they pass through disparate cables, telephone offices, etc.).

As an example, an installation in downtown Boston – there are two telephone company central offices which serve our building (this is fairly common in medium to large cities), Franklin (St.) and Harrison (St.), as well as a variety of leased line providers – ATT, NYNEX, MFS, Teleport, etc. This is in addition to radio and satellite receivers, which can also be used to receive data feeds.

If we bring in our primary leased line from the Franklin St. central office, we will likely back it up with a dial-up link, served from the Harrison office. The second set has the reverse configuration.

We generally ask for, and, if necessary, pay for a separate telephone cable entry point to the building which comes from the second Telco central office. It is surprising how often street repairs cause cable breaks. A cable break on the only feeder cable (or both cables which run side by side) into our building can easily knock out trading for a very long time even days.

Feed Redundancy

There is a distinct possibility that we will experience an equipment failure in the feed system, to mitigate this we duplicate all the feed equipment and insure that we have divergent routing – perhaps to another office of the feed vendor (i.e., no single failure takes down our data source). Depending on your business needs (i.e., risk tolerance and budget) the "secondary" system can be used to provide half of your capacity. You would then operate with lessened capacity in event of a failure – this saves some money, and creates some risk. Once again the business people help guide your hand.

The Teknekron Software System that is uses on our example floors does an automatic roll-over when one feed machine dies (and has been somewhat augmented to provide automatic failure notification to the system staff).

If a particular feed dies during the day, the roll-over occurs automatically, and the traders continue to see real-time data, as if nothing happened. A real sign of success is that the traders never notice these infrastructure failures. However, even though data is still available, it is important to restore our failed feed as soon as possible. I like to use the metaphor: "When one of your legs is broken you do not necessarily have a mobility failure, however it is extremely important to repair it ASAP. Especially in case the other leg breaks."

The failure could be caused by any number of things, but it is not uncommon to have a leased line failure (repeat, not uncommon). If this is the case, it is usually a simple matter to go to a dial-up modem, backup solution (hence dial backup).

Checkerboarding

If you take a look into the trading floor infrastructure you will see that the floor is laid out, at minimum, in a checkerboard fashion. Take a map of trading positions the floor (grouped by business unit is best) and color alternate desks light and dark. The dark colored squares are served by one complete set of equipment and links, and the light squares by

another (including cabling, power, hubs, servers, links and as much else as is within your budget and/or practical).

In the case of a catastrophic failure which takes down an entire color of squares, those traders can always look on with their neighbors. We're particularly fond of reminding the telephone group that long phone (handset) cords are part of our backup strategy. If a trader's workstation goes down then they can often continue to trade as long as their phone reaches to the next desk.

Note – it can be quite difficult to avoid all single points of failure – often you cannot get building management (in leased facilities) to insure that your cable runs (from the telephone company points of entry) come through different risers in the building (risk – fire/damage in the riser). There may be only one UPS, or one elevator, perhaps only one entry door. The failure of any of these things can make it difficult, if not impossible to trade.

Keep in mind also, that through floor expansion you will want to maintain your checkerboard layout. It's not always as easy at it may look at first glance. Our best advice: layout the entire checkerboard at the beginning and fill expansions into the original scheme.

Market Data

One of the primary types of information that traders base trading decisions on is market data. Market data consists of real-time prices, news and analytics driven by both intra-day and historical data. The sources and delivery mechanisms for this data are critical to both it's accuracy and reliability. In order to provide the configuration best suited to the business needs it is often necessary to understand the a number of aspects of market data: sources, costs, and data formats.

The primary sources of market data, for exchange traded instruments such as equities, are the exchanges themselves. For the over-the-counter traded markets of foreign exchange and money market instruments. The primary sources are dealers and inter-dealer brokers.

Large institutions may bring direct exchange feeds in-house. However, most firms contract for the services of a market data provider. Market data providers act as resellers of market data, consolidating the data from many sources, and re-distributing the data based on an institution's interest. The market data providers collect information from exchanges, dealers, inter-dealer brokers, inside sources and news services, and deliver a consolidated subset of this data to trading firms via digital data feeds.

The feed handler machines are on their own private subnet (feed network) because the universe of information available from our data providers is so large, and also to provide a measure of isolation. The feed vendors provide catalogs of the data they have available, some of them are distinctly reminiscent of the Manhattan, NY telephone directories (i.e., large).

Different market data providers transmit data at differing speeds. While the current line bandwidth used by these providers runs from 9.6Kbps to 128Kbps, the majority of these feeds are delivered at 56Kbps for broadcast feeds, and 19.2Kbps for interactive feeds. Most market data providers transmit their feeds using some type of data-compression and/or encryption scheme.

The data providers charge a handsome sum for their services, and they have a vested interest in making certain that their subscribers receive only the data they contract for, as well as keeping their cost of doing business as low as possible. In many cases, the feed vendors have a regulatory incentive to protect their raw data feeds as they are specifically liable to exchanges for user fees (often based on the number and types of people looking at exchange data).

Market data provider feeds fall into three categories from a record format perspective: record-based (elementized), page-based, and hybrid. The management of each type of feed requires specialized software.

Page-based digital data feeds deliver data on a "screen-by-screen" basis. To maximize communications efficiency, pages are transmitted as compressed text segments with each message header denoting the screen coordinates at which the data should be displayed. Control sequences that determine display characteristics such as color can also be contained in these text segments. Consider the daily foreign currency quotes from a newspaper and you will have an example reminiscent of what traders are seeing updated on a real-time basis.

In order to maximize the efficiency over the limited data link bandwidth, only the part(s) of the page which are changed need to be transmitted. These deltas can be applied to the "base" page producing the current completely data display.

Page based digital feeds are a carry-over from the traditional "green screens" of the past – where "traders" would watch the markets on one formatted page of an old terminal (with green phosphor). Today, while there are still green screens installed (and, alas, new ones being installed even now) – various methods are used to provide the similar functionality.

Some of our vendors transmit the basic screen to our system when it is requested (or subscribed) by a user. This is the mark of an interactive feed, where the user requests are sent up to the vendors equipment (i.e., outside of your site). There are others that provide a broadcast feed where they send

literally all of the information they have available (in a highly compressed form), and the local system(s) maintain all of the information at all times.

There are advantages and disadvantages to both methods. The interactive system tends to be a more complicated overall system, and new requests during periods of heavy activity may suffer a delay in processing.

The broadcast feeds have a simpler overall system design, but since there is a limited broadcast bandwidth, they often send base pages, and other "core" database information during periods of low activity. During the day, they are sending either price updates (whereupon the local equipment can expand those into formatted pages), or page updates. The disadvantage of this is failure recovery time. It often takes 48 or 72 hours to totally refresh the local database in your machine. Which is a maddeningly slow recovery from a system failure. Likewise, there can be a lag or lost data during periods of high market activity, as the broadcast bandwidth is limited. Since they are sending ALL the data they have that you may possibly request, some data of interest may be delayed or dropped.

While you might think that it is permissible to drop old prices for an instrument (say the asking price for IBM), if you don't have time to send them all down the line – often traders want to see ALL of the prices, even those they can't execute on. It's also potentially the difference between a price graph with a reasonable slope, and one which is dramatic.

In order to provide programmatic access to the data distributed via page-based feeds, it is necessary to apply an additional processing step, often called shredding. A shredder, or parser, will unformat a page breaking down each line of data into individual elements. Page shredding adds a small time delay before the data can be put to use. If the market data provider changes its page format, you will need to update your shredder. Unannounced page format changes are maddeningly common.

The product of page shredding is a record-based, or elementized, data set. This type of record format is suitable for use by other programs, where it can be used for trading and analytical applications. These elements could be stock prices, exchange rates, bond yields, currency quotes, news headlines, etc.

Record-based data feeds, also known as elementized feeds, deliver data according to a defined set of record formats. The main benefits of this type of feed are: no shredding is necessary; the data is presented in a known format regardless of its source; and updates can be processed without the delay, or potential hazard of shredding.

Real-time elementized market data is used as input to a number of decision support systems. These systems include real-time spreadsheets, for instance performing simple calculations such as the conversion of bond prices to yields; screening and analysis programs, which generate alerts based on certain events such a IBM trading above 103; and risk-management systems, which interface the real-time feed to a position management system.

In addition to real-time feeds, the typical trading floor will integrate numerous other data sources into its applications environment. These other data sources will supply data that, while still critical to the trading decision process, does not change as often. These data feeds may include batch downloads of holdings, historical databases and databases of SEC (or other regulatory body) filings by institutions. The data can also be delivered via FTP, over dial-up lines, or on magnetic media.

Data Distribution

Now that we understand the sources and record formats of market data, we must investigate what is done with it after it is delivered to the trading floor computer room. This brings us to the market data distribution plant.

At the heart of the market data distribution plant are the feed handlers and ticker plants. Feed handler software generally runs on a dedicated computer, sometimes with specialized hardware (i.e., high speed serial interface). The feed handler manages all communications with the market data provider. The feed handler is attached to the market data provider's system by means of a serial or dedicated LAN connection. It is attached to the feed network on the trading floor side.

Also sitting on the feed network, are the real-time database applications, typically called ticker plants. These ticker plants maintain a memory resident database of current values and subscribers. Current values are the latest values for a specific record or page transmitted on the feed. A current value might be the most recent image of Knight-Ridder page 45000, or the latest bid and ask prices for IBM. Subscribers are applications running on the traders' workstations, analytics, and sometimes the software which drives the large wall-displays which are so popular in trading rooms.

There are usually real-time database machines (which store, at minimum, the latest prices for various instruments), and these are also duplicated in (at least) a fault-tolerant pair. The real-time database machine is necessitated by the need to supply the latest quote for a particular symbol – essentially the instant it is subscribed to.

Since you must also cover the possibility of a machine failure the realtime database machines are duplicated. So, it is not a requirement to use disk mirroring or RAID.

The trader workstations reside on a trading network different from the feed network. This requires

the ticker plant to be dual-homed to the feed network and the trading network.

Processing mechanisms for the feed handler vary according the the type of feed, page-based or record-based broadcast/interactive. The main difference between page-based and record-based feed handlers is in the methods they employ to request and process data. In a page-based feed, requests are submitted as they occur. This usually, but not exclusively, translates to a user "punching up" a given page on a trader workstation. Requests can also be transmitted when the feed handler is started as a means to immediately populate the database with the most frequently requested pages. The feed can also be told to mark these pages so that they are always fixed in cache. This is a method of ensuring that when the maximum page capacity is reached, these pages will not be swapped out to service new requests.

In a record-based feed, records may be selected via a filtering criteria which might include instrument type or exchange name (i.e., all NYSE stocks). These feeds operate at speeds equivalent to the delivery of hundreds or thousands of records per second. The This feed handler selects the required data and transmits in internal format onto the feed network. The values can then be inserted into the current value database of the ticker plant.

Recovery methods also differ between record-based and page-based feeds. As the page-based feed is interactive, refreshing the current value database is a matter of re-requesting pages. This is not an option with the record- based feeds. A record-based feed would need to receive a "refresh" (i.e., values for all symbols of interest), in order to re-populate the database with all instruments,

As with any other part of the trading system, the design of the market data distribution plant is required to include mechanisms for reliability and fault-tolerance. As a rule, for any real-time feed, we install a minimum of two feed handlers and two ticker plants. These four processes run on up to four different computers, are connected to the market data providers using divergent routing and back-up dial lines (or ISDN) using alternate facilities.

The software installed on the example floors, enables us to run the two feed handlers as a fault tolerant pair. We also run the ticker plants as a fault tolerant pair. As with the ticker plants, the feed handlers operate in primary or secondary mode and maintain this relationship by use of a time-stamped heartbeat. In addition to taking over as the primary feed handler (or ticker plant) upon failing to receive heartbeats within a time period, the new primary feed handler will re-broadcast the data it received since the last heartbeat was received. This assures that no data will be lost for the time needed to "fail-over".

On the client-side, between the ticker plant and the trader workstations, Teknekron generally delivers data to the desktop using UDP. A reliable delivery mechanism is built into the data portion of the packets which contains a sequence number. The UDP broadcast is forwarded to all IP subnets on which client workstations require access to the data. Cisco routers have an "IP-helper" feature which helps implement this.

Trader Workstations

The term trader workstation refers to the computer system that sits on a trader's desktop. The trader workstations we install on these floors are Sun Sparcstations; at present, running SunOS. The windowing system is the X-Window system with a virtual window manager. These trader workstations offer a cost-effective scalable platform; capable of real-time display and processing of graphics and compute-intensive applications.

This platform is a conservative choice, from the viewpoint of its being widely supported and proven. Yet it affords us the flexibility to take advantage of emerging operating system and network technologies.

Servers

In the "computer room" you will find the usual NFS, or other file servers; but, generally speaking, all the critical disks are either mirrored or RAID protected. In some installations, the file server itself is backed up by either a hot or cold spare. Depending on your needs, it is also possible to build redundancy into the CPUs, or file servers (using specialized hardware and/or software).

The "back-end" machines are also Sun SPARCS, running mostly SunOS, which follows our preference for proven and reliable technology. You may see occasional specialized servers running Solaris (e.g., SparcServer 1000/2000 used as a Sybase machine). However, in general we stick with technology which is proven and known reliable (at this writing, more significant Solaris transitions are taking place).

Network Topology and Cable Plant

Similar trade-offs to the decision regarding choice of trader workstations come into play when we design the network topology. We can't afford to put "inexperienced technology" into production. So, when it comes to the network we try to position ourselves to take advantage of emerging technologies without having to undertake a major infrastructure retrofit.

We start with this perspective and also the need to support the special network needs of trading applications, including redundancy and protection against single points of failure. We must also keep

in mind the need for expansion, such as additional market data feed services, additional protocols, and even remote sites such as branch offices.

The example trading floors are currently using 10Mb Ethernet. It is extremely important to characterize the LAN traffic generated by trading applications including the heavy UDP broadcasts in order to determine an appropriate number of trader workstations per subnet/segment. This also helps when setting alarms for network utilization rising above/below the nominal band, which is often an indication of impending problems.

Shared Ethernet is also potential problem. While it is a technology that makes it easy for multiple machines to communicate, the latency of the network is impossible to predict. Some installations have multiple Ethernets running to each workstation in order to minimize delay and the potential that an unintented network "event" (i.e., user FTP of a large file) will block time-sensitive market data. One of the Ethernets is dedicated specifically to transmitting market data, which effects a hard partitioning of network bandwidth for real-time traffic.

We also wanted to protect the trader workstations from network problems that could be isolated to a specific subnet, such as a hub failure or a broadcast storm. To this end, we installed hubs (connected by routers or bridges – not wire or fiber) with redundant power supplies. This protects not only against a failure in a power supply but potentially a power circuit failure as well. If your device has redundant power, be sure to plug each power cord into a different power source, or at least a different circuit.

Hubs are often given short-shrift by networking people, as they are "uninteresting" devices. However, to us the failure of a hub can take down a large number of traders – so we treat them with respect.

The cable runs to the trader workstation, consist of home run Cat. 5 UTP – which we consider fairly future-ready. No two adjacent desks are connected to the same segment. In the event of a segment-specific problem, the affected trader can "look on" with a neighboring trader. This is the hub view of "checkerboard" or "salt and pepper".

Whether to run (or even use) fiber at this time we find is largely a matter of cost or taste. On average, you want move or do a major technology upgrade of trading floors every 3-5 years. You can spend money today in an effort to delay the major upgrades, but your success is a matter of how well you can predict technology and business change.

Many times installations will attempt to "future proof" their site. Some designers choose to preinstall significant additional cabling (or fiber), and others install a conduit to each desk. This makes it relatively easy to place the appropriate future cable system in place. The majority of trading floors that I've seen are built on raised tiles, this aids future installations, and in often required by the sheer volume of cable running to the desks.

Technology which delivers video over Cat. 5 UTP cable, whether through the use of passive (baluns) or active electronics is commonplace. This can be a major consumer of Cat. 5 "pairs", and should be considered as part of the cable plant design. Several vendors of systems which require video delivery and previously required proprietary cables, now affirm these systems, or install them themselves. This can be a major consumer of Cat. 5 pairs to the desk and should be considered as part of the cable plant design.

Keeping proprietary, and unique cables out of the environment is extremely important. Not only from a "cleanliness" and cost standpoint but also to assist future moves and changes. To move a trader from one position to another, when all the wiring is structured, involves merely disconnecting the equipment at one location, connecting at the other, and arranging for the appropriate cross-connects in the cable plant. Contrast this to special cables, which will need to be run to the new location, irrespective of whether they are removed from the old one.

System and Network Monitoring

The need for system and network monitoring, in this environment, can be daunting. The management platform employed must be capable of several functions in order to be effective. In addition to detecting and reporting problems, it must also be capable of utilization and performance trend analysis and sometimes business-related functions (i.e., charging for usage); and it must be possible to incrementally expand it.

We use a combination of Sun Net Manager, application-specific tools and other vendor products. We monitor the health of every computing device on the network. Monitoring elements act on events to change the color of component icons and beep or send e-mail to the operations staff. We have found that the most useful notification device is the alphanumeric pager, which is driven automatically from the monitoring system. [It is possible to have redundancy here whether via multiple lines to the beeper vendor, multiple beeper vendors, or even multiple people receiving the same message.] The propagation delay of a message from our system to the beeper itself, is sometimes a problem. We have tuned our system so that only critical messages are sent during sleeping hours.

If you have a particularly large floor, or a somewhat inexperienced systems staff it can be quite helpful to scan in a floor plan of the trading area. You can then have the appropriate desk position flash when there's a problem. This can saves significant time tracking down the user.

In addition to checking connectivity and resource utilization for every trader workstation, we also monitor application processes and scan log files for clues that there may be an impending problem. We often inform a user that there will soon be a problem with his/her workstation unless we can restart this application. We apply the same vigilance to monitoring the health of our servers, filesystems, etc. The amount of mileage you get from a very small amount of code that triggers various (warning) actions based on matching particular regular expressions from a log file is surprising.

When used heavily with real-time updates even some of the best known/loved software develops memory leaks and other anomalies [some favorites are spreadsheets which are driven to recalculate by the change of a symbol's price (i.e., IBM trades up or down) – this, in turn, can cause significant amounts of recalculation, and after a many commercial spreadsheets show their bad manners). One or two applications in this state, and your workstation starts to thrash it's disk. All applications run slower, and real-time applications are delayed.

Routers, hubs, ethernet switches, and multiplexers are all monitored for performance according to their specific function. A high count of dropped packets on a router interface could point to an looming problem that could have a significant impact on trading applications and delivery of real-time market data (circuit failure?).

Application Wrappers

Another thing that we often do is "wrap" critical processes in a shell script, or other device. This serves not only to notify on process failure (i.e., core dump), but also gather any potentially useful information and restart the process. You might be inclined to think that if a process dumps core it should be left dead until repaired. However, if the process works most of the time or at least a fair amount of the time, and it produces data which is essential to trading – you must restart it during the trading day. The true repair or upgrade can be scheduled during off hours.

Systems Staff

No matter the level of sophistication of the System and Network monitoring infrastructure, it is most important to have an operations staff that is capable of responding swiftly and decisively to resolve problems. This is the greatest challenge in supporting the technology of a trading floor.

Our number one focus is (near) 100% reliability during hours of interest. In some markets there are defined "trading hours". For instance, 10AM to 4PM for New York Stock Exchange (NYSE) regular trading session. Any failure during these hours can be extremely expensive.

We're quite used to hearing "this just cost me 1 million dollars" (converted to the currency of your choice), it seems our traders always have a million, or so on the line, and a system failure of any proportion always has that price tag. If indeed this is the case, then it should be relatively easy for us to justify additional redundancy to the business managers.

The rule of the road is accurate timely data, or no data at all (preferably an indication of failure). Because of the complexity of the systems, and the number of external feeds (providers) – constant vigilance is required. You might be tempted to think that in a situation where you don't have the current price for a symbol that displaying the last price is OK. Actually that is bad data (stale data), and another rule is Bad data is worse than no data.

If a trader doesn't have a price he needs, he can pick up the phone or use some other piece of equipment to find it, but if you're giving him an old price, which is, in fact, a bad price – then you are actively working against him. And your technology is, in fact, worse than having nothing at all.

It's unfortunately all too common to lose a particular data feed during trading hours. This can be the effect of any number of failures – from a wire being pulled or loose in our cable plant (it's amazing the number of times electricians working in your building will accidentally interrupt service) – to any number of feed vendor infrastructure problems.

The nature of the problem determines the action we take to restore service. If we lose a leased line (or some of the data comm. equipment involved in that circuit), we can immediately go to a dial back-up scenario. In general, we will ensure that we have dial-up access to an access point other than the one our feed comes through. An example is a leased line which originates at the vendor office in New York, then our dial back-up number could be at the vendor's home office in Kansas. Or, quite frankly, anywhere in the world where a direct dial modem connection can be made. The volume of trading on our floor, and the potential exposure from faulty or missing data often quickly covers the cost of a call ANYWHERE.

There may come a time when the cost of the call becomes prohibitive. However, the mere fact that you have the ability to restore service over a dial-up (perhaps long distance) connection, means that you have the ability to make this cost/benefit decision.

Remember that we have spent a great deal of time engineering divergent circuits. In the unlikely event of a problem at one of our Telco central offices – a fair number of circuits on ONE SIDE of our redundant plant can be affected. We can initiate dial backups on all of those feeds and be back to fully operational in a short time. [Our dial-up lines are engineered through a different CO than the

leased lines.] A very similar circumstance is a cable cut on the main feeder cables into my building. Once again, we can restore full service from this catastrophic event very very quickly (given multiple cable entry points to the building).

Depending on the nature of the problem – we contact the appropriate people to restore (normal) service ASAP. It sometimes requires a keen political sense, as well as good technical grounding (in a number of disciplines) in order to assist vendor personnel of varying abilities to resolve our problems quickly. We have often walked various vendor personnel through trouble shooting procedures over the phone. Procedures one would hope were part of their basic "tool kit".

During an outage, the keys are to find the malfunctioning subsystem and patch around it as soon as possible. Fixing blame, whether it be organizational or personal, is an activity that is best left to the post mortem. We are of the opinion that once a problem is resolved the appropriate follow-up is to document what happened, how it happened and what steps are being taken (have been taken) to insure it never happens again.

After we've restored service, we have the latitude to schedule repair or replacement of the circumvented, failed device(s).

The necessity of near 100% service during trading hours often forces us to fix problems twice. The first time, during the heat of the trading day is simply enough to get things operational again. Later, you schedule an actual repair or upgrade.

We're performing tasks which are mission critical, often with technology which is not designed for this purpose.

Elimination of NFS

In our quest to remove as many single points of failure as possible it is quite common for us to remove dependencies on NFS. What that means is, as far as file access is concerned, making each trader workstation stand-alone.

In this age of multi- GB internal disks, the hardware itself offers few limitations. However, the prospects of keeping a network of 200 300, or 1000 workstations all up to date with the latest copies of applications (in some cases, varying the version depending on the configurations needed by specialized applications used by this trader) can be daunting.

Not only keeping the workstations up to date, but keeping track of what versions are running on each workstation – in case a new software "roll-out" causes a problem. Also required: a quick and reliable way to roll back to a previous version.

One might think that the need for rolling back to a previous version points to a certain lack of Quality Assurance in the process. And frankly, that's true. However, given the complexity of the environment, and the dependency on real-time data which is not easily characterized (were it easy to characterize the data (e.g., predict the future price of IBM) you could be making big profits in the markets) – it is VERY difficult to test applications thoroughly.

That's not to say we don't QA things, it's just that we have found it to be prudent to always be ready to roll-back to the previous version of a newly installed application.

Keep in mind that the amount of money at risk on a daily basis easily dwarfs our paltry salaries.

Automounter

While automounter provides a valuable service, it is not often welcome in a redundant configuration which uses network file service. The reason is that you cannot reliably determine which of the multiple potential servers for a filesystem a client will bind to. If you take a salt and pepper scenario, and you have dependencies which cross colors, you may have made the system less reliable than if you had done nothing at all. Certainly you have not been able to guarantee that a particular client machine depends on nothing which it's neighbor requires.

NIS

Because the failure of a centralized NIS server can be disabling, we often have each client machine act bind to itself (NIS slave) or remove NIS dependencies entirely. The update process, which usually does not take place during "production hours", is not trading critical, hence propagation from a central host is acceptable.

Hardware configurations

For critical servers where the software is not equipped for real-time redundancy (or fail-over) we often keep a stock of backup machines. This can vary from one "cold" backup machine per critical server to a smaller number which serve to backup many. Here the rule of the day is to equip the cold spare with the right amount of memory or the max. of all the machines you are covering, and NO SERVERS have internal disks.

Given a cold spare with the right internal configuration (memory, adapter cards), and well laid out, well labeled cabling – you should be able to swap out a failed CPU in a few minutes, plus reboot/restore time.

This implies that the software running on this server is not hostid specific. It also implies that your computer room is designed well enough that you can easily access your spares (as necessary) and the cables of the failed machine.

The best intentioned software licensing scheme can often wreak havoc on the best intentioned backup scheme. Our best advice – test your backup scenarios regularly.

What do we do when a user workstation fails? Generally speaking, we have a stock of identical spare machines on hand. The user's workstation is swapped for a spare, they are back in business, and we have a broken machine to deal with at some time in the future.

Computer Room Layouts

We have fairly strict rules about the way hardware and cabling is placed in our computer rooms. Besides the normal cleanliness and orderliness comes:

0) Physical security limits access to the computer room, and specifically, trading critical elements – to those with proper authorization (and training).
1) no machine will be directly placed on top of another independent machine. If you do this, then during production hours, the machine on the bottom will always be the one to fail. And you will be faced with the uncomfortable need to take down the one on top in order to fix the other (thereby adding a problem in order to fix one).
2) no machine, and it's redundant pair will be directly adjacent to each other. This affords us the luxury of possibly surviving some significant disasters, i.e., water leaks, VERY well-contained fires, mechanical failures.
3) no machine, and it's redundant pair share the same power circuit. In cases where the size of the installation warrants, and budget permits, two independent UPSes are installed.
4) All critical equipment is protected by UPS, and UPS protection is tested. We're particularly fond of computer rooms where only UPS power is available. But in instances of limited UPS battery life (and no backup generator) it is often helpful to judiciously apply UPS power. In the latter case – you can often save a significant amount of power, and hence extend run-time on battery by removing non-critical monitors from UPS. Multiple power feeds with divergent routing, is not unheard of, in our business.
5) All cables are secured. Whether this means the screws on cables are locked down, or ethernet AUI retaining clips are used. Modular/universal power cords have a tremendous affinity toward working themselves loose at the back of the machine, so we make sure they are not under tension. Some manufacturers have fitted their equipment with a retaining clip, and we applaud them.
6) All cables are managed. Excess cable is either eliminated (i.e., you get the right lengths), or secured, where necessary.
7) We color code things wherever possible, and label EVERYTHING.

It may seem pedantic to spend this much time and energy on something as seemingly boring as a computer room installation, but during an outage, a few seconds saved in restoring service can be a LOT of money.

Given Murphy's law, the person closest to the computer room during an outage is the one with the least experience. So we're also trying to make it easy (almost trivial) for them to work on the hardware.

Yes, sometimes even trading support people go to the bathroom, or, god forbid, take lunch.

License Managers

Quite often, vendor software uses some sort of network license manager. Generally today's crop of software found on trading floors (in fact PERMITTED on our trading floors) uses a fault-tolerant and redundant license manager scheme. We routinely run redundant license managers for all of our licensed software, and there, the remaining exposures come from network partitioning (so that a quorum of managers cannot be achieved) and the fact that keying a license manager to a particular hostid will often be problematic if we execute a quick changeover to a cold back-up CPU. Oftentimes you can work with software vendors in order to alleviate concerns you have in these areas.

Production vs. Test

As the technology marches on, we must stay on top of it. This affords us the need to test and experience a wide range of technologies. After characterizing and evaluating new elements, we sometimes put them into production in a controlled way.

Introducing a new piece of hardware is little different from new software. Even the best testing in the world is only a simulation of the actual environment. Hence we are generally prepared to back out failing "upgrades".

However, one place where we are not always successful is in user developed applications. The TSS system discussed here give users/traders the tools to develop their own analytics and applications, and "publish" new data elements to the floor. While this is generally a good thing, and to be encouraged, if not carefully managed it can lead to subversion of the redundancy/reliability design, and unexpected dependencies on system elements considered non-critical.

We generally consider the failure of an individual trading workstation problematic, but non-critical, in the sense that the impact to the floor is

well contained. However, if that failed workstation runs a spreadsheet application, or some other custom analytic which is publishing to the floor, and many traders are dependent on it, then you have a problem.

Our best approach to this problem (short of discouraging innovation from our users) is to carefully educate them as to the meaning of test and production. If there are applications developed by our users which have a need to go "production", then we take control of them, add appropriate redundancy, run them on a server machine in a controlled environment, and finally institute strict change control.

Most spreadsheet programs are woefully inadequate in the change control area, and given that vendors encourage users to develop large business critical applications (i.e., financial models), change control is a necessity.

Moves and Changes

As if keeping one of these floors operational weren't a difficult enough problem – there comes a time in every trading floor's life when a move or major change is necessary.

Moving a trading floor has similarities to moving other production environments, with the added factors of many outside data feeds, very short period of allowable downtime (i.e., one weekend) and absolute criticality for 100% uptime.

Of course, it takes a number of years to outgrow the present quarters, or amortize the original expense, and over those years sins of omission tend to mushroom. We can't exactly blame our predecessors, few of them ever imagined that their "quick and dirty" installation which was meant to be temporary ("we'll go back to fix it later") would become a permanent part of the production environment, and in fact, critical to the trading process.

The process of moving a trading floor can be likened to "putting the left-over spaghetti back into the box". Basically, we need to completely understand the present environment (IN ADVANCE), design the destination environment to at least duplicate it, and then move in a manner with well understood and controlled risk.

This is often a case where spending some money makes it possible to lower the risk. There's often a fortunate correlation between the size and importance of the installation and it's profitability, hence a large a critical installation may well be amenable to spending money in order to reduce their risk.

The tolerance for risk is highly dependent on the organization, but in general it is common practice to pre-install all the infrastructure equipment. Each feed (all the redundancies), each server, all the hubs, routers, etc. If you are very successful then all that is required on move day is to move the users and/or their workstations.

Of course, the possibility of breakage during the move exists, so you prepare yourself with a number of spares, whether they be prebuilt for each configuration, or prebuilt with the "base", and a facility for quickly adding the required special pieces.

It takes a tremendous investment to prebuild all of the infrastructure, and this, again, is a demonstration of the dollars at risk, and potential profits inherent in the trading game.

If you are clever, or fortunate, you can often devise various ways to optimize on the amount of equipment you must buy new for you new environment. Some people try to align moves with the upgrade of various technologies, where they install the upgraded technology at the new site, and deinstall the old site to the scrap heap. This sort of "upgrade during the move" is highly dependent on your organization's tolerance for technology change risk.

If all goes extremely well on move day, you may find yourself in a slightly sleep deprived state, answering questions on how to use the new phones, and where the plumbing infrastructure is located.

Scheduling Upgrades/Changes/Outages

It is highly unlikely that you will be executing changes or upgrades during the trading day. In general, you will do this sort of work off-hours. We tend to like to start these activities just after the traders leave (if on a weekday) – in order to give ourselves all night to fix any problem/s that may occur.

Here, again, is a place where your organization, and the business influences the amount of risk that is permitted. In some shops NO changes of a substantial nature are allowed during options expiration week. That means that you need to keep more on your calendar besides your friend's birthdays, and the next technical conference.

The technical risk of change has to be balanced against the fundamental business need for improvements. If your financial engineers come up with a new, faster or better algorithm for computing an analytic – then there is a potential business gain for pushing that out as soon as possible.

This is in fundamental opposition to the operation manager's goal of limiting change (and hence, hopefully maximizing reliability) – and the opposing views must come to closure.

Support People

What kind of people do we hire to run trading floors? Well, to say that they are extremely bright is

a given. It's hard to characterize them simply, because the job has many facets.

On one hand, an exceptional view of the trading plant from a conceptual level is essential. That helps the person quickly home in on the component or components that may be causing our problem. It's not essential that they know each and every piece of the technology, but if they are not expert – then they must be able to help (manage) the experts in resolving the problem(s).

Another given is people who are fairly tireless. In this day of limited profits/budgets for Wall St. firms (as well as the world-wide "streamlining" of corporations) the staff available is going to be limited.

Since it is quite common to be doing floor support during the day, and upgrades on nights or weekends, a willingness to work more than a "standard" work week is often essential.

We also like people who are cool thinkers under stress. There are times when various people are describing a problem in an, ahem, excited way. It requires a cool head in order to filter the emotional from the technical and actually resolve the problem.

At times, just like in other environments, problems which are non-technical come our way. Probably one of my ubiquitous favorites, and one which really does deserve as immediate a response as any other – is the coffee drenched keyboard. We keep a few spare keyboards handy just in case of this (all too common) eventuality.

It also doesn't hurt to have a sense of humor – in fact, that came to mind when asked by a major Wall St. firm what the ONE most important quality in a floor support person is (the others being well known, and more obvious). Second would have to be the ability to build things quickly out of whatever parts are available (reminiscent of the American Television show MacGyver).

Acknowledgments

Some of the material presented in this paper was garnered from discussion and collaboration with Alan Kadin, Fidelity Investments, and Robert Suyemoto, formerly of Fidelity Investments and a consultant to Bank of Boston.

Author Information

Mr. Lipson spent numerous years designing and implementing systems software, LAN and WAN protocols at ComputerVision, BBN Communications, BBN Labs, and various consulting clients. In several years at the Open Software Foundation he worked on OS internals, including OSF/1 and OSF/1AD, international standards and DCE. The last several years he has consulted for financial services institutions including Morgan Stanley (US), Fidelity Investments and Bank of Boston. At present, Mr. Lipson is a consultant to Morgan Stanley Japan Ltd. where he manages the distributed systems and market data groups in the relocation of Tokyo offices and trading floors. Reach him via email at srl@kr.com .

Morgan Stanley's Aurora System: Designing a Next Generation Global Production Unix Environment

Xev Gittler, W. Phillip Moore and J. Rambhaskar – Morgan Stanley

ABSTRACT

The challenge: To come up with a distributed systems environment that would allow Morgan Stanley to centrally manage tens of thousands of systems spread out over more than 30 offices on virtually every continent on the globe in a fully production fashion.

The solution: The Aurora System.

History

Morgan Stanley is a global investment banking company. The day to day business activity of the firm depends in a large part on the stability, reliability and functionality of its technology. The firm trades on most exchanges around the world, and as such, there are very few hours in a week when trading activity is not occurring somewhere on our networks.

Until November 1993, Unix computing activities within Morgan Stanley were divided into two distinct groups. One business unit within Morgan Stanley, the Fixed Income Division, maintained its own computer management staff, called the Fixed Income Research (FIR) group. The Fixed Income Division required significant computing capacity, and, more importantly, realized and encouraged growth and use of equipment to its fullest potential. FIR introduced Unix systems in 1987 and by 1993 had approximately 1500 systems, 90% Sun and 10% IBM. FIR had nearly complete authority over all computing related decisions.

In contrast, Information Systems (IS) was responsible for the vast majority of Unix systems throughout Morgan Stanley, reporting to many different business units with different needs and desires. In some instances, IS was allowed to suggest technological solutions to problems and implement those decisions. In many cases, however, it was presented with systems to manage with no initial consultation, and any suggestions, especially those that required any additional expenses, were overridden by the business unit. IS introduced Unix in 1990 and by 1993 had approximately 3500 systems, all Suns. IS and FIR rarely cooperated.

In November 1993, FIR, IS and all other computing groups merged into the Information Technology (IT) department, consolidating all computing related activities under a single organization with significant decision making authority. After the merger, in mid-1994, the top distributed systems engineers from both IS and FIR formed into the Core Infrastructure Group (CIG). The mandate given to this group was to create a common distributed systems technology platform for the firm. The CIG had the unique opportunity to dream up the ideal Unix operating environment for our needs and make it happen.

Design Goals

Up until (and through) the merger, our network expanded via the installation of new workstations at a rapid pace. As a result, the system components that we had in place were starting to show signs of strain. For example, our file system distribution was becoming unwieldily, taking too long to complete and systems that should have been identical were no longer in sync. Additionally, replicated copies of the system data accounted for almost 40% of all used disk space, which was far too much overhead.

The pre-Aurora environment was functional, and under ordinary circumstances we could have made it last a few more years without a major overhaul. However, because the systems would undergo a major upheaval in any event as a result of the merger, we decided that we might as well go whole hog and design the perfect system.

In addition to remedying shortcomings of the pre-Aurora systems, our perfect environment needed to address the following issues:

Scale – A major criteria of the environment the CIG developed was its adaptability to scale. We define scale somewhat differently than most, based on our diverse physical requirements in our various locations. Morgan Stanley is not set up in a traditional campus setting. We have over 5000 systems distributed globally in over 30 locations. Our largest offices have over 1000 systems, while the smallest offices have fewer than five. The bandwidth between offices varies from 56Kbs to hundreds of Mbs. The largest offices have hundreds of support staff, while the smallest have no support staff.

Based on this infrastructure, any system that we designed not only had to scale upwards in the traditional sense, but it also had to scale to support the smallest sites as well. This requirement means it must fit on small systems, use less bandwidth and not depend on many different servers to support the environment.

System Usage – Given the nature of our business, even minutes of down-time are unacceptable. Our systems must be operational 24 hours a day, 7 days a week. Except for a few hours a week, trading is always going on. Additionally, we have a fairly static user environment. Users log on to particular workstations, and typically stay logged on until the system reboots. Running jobs and screen configurations are painful for the users to restart. As a result, system reboots only occur on the order of months.

Because our environment must run with zero down time, in designing Aurora we tried to avoid products that only ran in university or research labs. Every part of the environment that we use must either be commercially supported, or we must support it internally. It is important to realize that we were not designing a research project. Project Andrew, for example, which was a research project, took three or four years of determined effort to stabilize after the initial rollout. We did not have this luxury.

Global Usage – Many of our users travel extensively between offices for their work. We designed the system based on the concept that wherever a user went, when she sat down at an available workstation and logged in, it would be her environment. If a trader in New York jumped on a plane to Singapore and logged in, all his files, the programs that he normally ran and the data he normally accessed would appear with no operator intervention and minimal performance degradation. There actually are regulatory and contractual restrictions on what and where users can access data and programs, however there are mechanisms other than visibility for enforcing these restrictions.

In order to manage a global environment of this size with the small number of people we have in our operations staff, we had to provide a single operating environment worldwide. The sacrifice that we made in doing so was to place restrictions on developers and business units about what they could place into the environment, and what they had to go through the operations staff to do. However, with proper hooks to allow for customization, we felt that this would not be an onerous burden in relation to the economies of scale that this scheme would provide us.

Design Focus

Rather than merely merging the existing FIR and IS systems, the CIG decided to provide a necessary level of interoperability for the short term, and focus on creating a new system from scratch, using the best proven technology available. We used the opportunity to throw out many of the bad features and unused functionality of the old system. We did not provide backwards compatibility by default, we provided it only if it was proven that it was required functionality that was not provided in the new system. We would take the best of the old systems and the best that the marketplace had to offer. For no particular reason, we named the system "Aurora".

In designing each aspect of the environment, we always kept in mind the 4 Rs: Redundancy, Reliability, Recoverability and Reproducibility. In addition, before we built something ourselves, we surveyed the market to determine if a product was available that met our needs. Unfortunately, in far too many cases, the products either were not close enough to our needs, or required significant local customization. If a vendor's product provided functionality that was close to meeting our requirements, we attempted to use Morgan Stanley's size, buying power and clout to encourage the vendor to modify the product to meet our needs.

Finally, it was important that the system we designed was abstract enough to support multiple hardware architectures, yet still provided an interface specific enough so that developers could create both system and user applications that run on all Aurora platforms.

System Design

There are four key components to Aurora:

1. Global Look and Feel (Global Desk)
2. Global File System (AFS)
3. Global Configuration Database (DSDB)
4. Global Homogeneous OS Configuration

Global Desk

Choosing the Window Manager

The change that was most visible to users was the replacement of the window manager on the desktop. Before Aurora, different groups were using different window managers, including olvwm, mwm, twm, tvtwm, fewm and others. In order to avoid having to modify the user's window manager again for a long time, we peered into the future and decided to go with the Common Desktop Environment (CDE), which appeared to have enough vendor acceptance behind it to win the desktop manager wars. However, CDE, as it was implemented at the time, did not have sufficient functionality for our environment. In particular, it had the following drawbacks:

- No support for multiple displays.
  In our information intensive business, our users need to be able to see as much

information as possible. As a result, a large percentage of the desktop workstations have between 2 and 4 displays.

- Inadequate Workspace Management
  CDE did not have a graphical window manager. We have users that actually make use of up to 60 workspaces. Navigating through windows is done via toggles that move either one window forward or one back. While this may be sufficient with a small number of workspaces, it is not adequate for a large number of workspaces.

In order to address these shortcomings, we partnered with TriTeal, the company mainly responsible for the development of CDE. At our encouragement, they added the following major features to CDE that we felt were required (along with many other minor mods and bug fixes):

- Support for multiple displays, including support for multiple control panels
- Graphical Workspace Manager, similar to the workspace manager found in olvwm
- Modifying workspace buttons, so that if a user has 60 workspaces, he doesn't need 60 buttons
- Enhancements to assist non-ICCCM-compliant applications

One of our key requirements with TriTeal is that anything they develop that is not explicitly Morgan Stanley specific would be rolled back into their product. We did not want to be stuck with a "consulting special" that would cause us to be unable to upgrade to new versions, and we did not want to have to support the product ourselves.

X Window Server

Until recently, we were using X11 compiled from MIT sources, rather than relying on the vendors to provide us with a working version. We did this because the vendors generally lagged behind MIT by a few years, and when they did implement a server, it was slower and buggier than MIT's. However, in the last few years, the vendors' implementation have gotten much better, to the point where they provide significant benefits over the MIT server. In particular, they provide real customer support, as well as support for all devices they sell and Display Postscript. As such, for Aurora we have decided to use the vendors' native X11R5 server.

Rollout

Because this was such a visible change, we decided to split the rollout of Global Desk from the rest of Aurora and implement it in advance of the rest of the Aurora changes. In doing so, we believed that when we converted users from their old IS or FIR system to Aurora, they would not even notice a change. We also took advantage of the window system rollout to upgrade all users to use a new standard profile system.

Profile System

The profile system was one of the pieces that we reused from the pre-Aurora environment. It allows us to set up users with standard sets of configuration files, and gives us the ability to easily update users' profiles and ensure that their profiles are correct. By giving the users hooks to allow local customizations, we are providing a commonality and standardization, while ensuring that users will not try to get around the system. The profile system works as follows:

There are a number of base configuration files, like base.profile, base.vuewmrc, base.envfile, etc. These files are common for all users. In addition, there are "model" files. These are typically set up by business function. The files are located in a central directory, and are set up via symlink to the users' home directories. This allows us to make global changes easily, without editing files in each user's home directory. For example, the "Governments Trading Desk" model might set up paths, environment variables and window system configuration to include all the programs that they use. The "Sysadmin" model on the other hand, might have all the "etc" directories in its path. In addition to the common files and the mode files, there is a .custom directory that may contain optional override files.

There are three commands available: installprofiles, checkprofiles and restoreprofiles. A user is set up with the installprofiles command. This saves all old profile information (which can later be recovered using restoreprofiles), and installs symlinks or copies of files from the central installation directory. The files that are installed contain references to other files in a model directory, and to files in the user's $HOME/.custom directory. The model of the user is determined by the contents of the $HOME/.model file. In addition to static files tat are copied or symlinked from the common location, there are "action files", that specify programs to run to populate other files in the user's home directory. For instance, the ".printer" file, which contains the name of the default printer, is generated by picking a printer based on the location of the user as obtained from the corporate database. However if that file already exists, the program assumes the user does not want it changed, and just leaves it.

The checkprofiles command compares the contents of the user's configuration files to the contents of the common configuration file. When a user calls the help desk with a problem, one of the first things the support personnel do is run a checkprofiles. If the contents differ, the user can reinstall the standard profiles, and check for the problem again. If the problem is gone, we have significantly reduced the problem set. In some instances, we may even tell the user that it is his own problem, and he must fix it himself.

This model provides a powerful resource for easily making global or departmental changes without having to run around and modify files in every directory.

Global Filesystem Project

Limitations of the Pre-Aurora Environment

Our pre-Aurora distributed filesystem environment is built entirely with NFS. We have gone to great lengths with the free-ware automounter, amd, to try to maximize the functionality provided by the underlying NFS technology, but it has, at best, several limitations:

- Redundancy/Replication
  Replication is obtained by using locally developed scripts which use rdist and track to force remote copies of "replicated" data to remain in sync with a master copy. Redundancy is obtained by configuring amd to mount key filesystems from one of potentially many sources. Regardless, when an NFS mounted filesystem hangs, it hangs. If you are paging off of an NFS mount, there is no transparent fall over mechanism to allow use of available backup filesystems – you core dump or hang.
- Building-wide shared namespace
  NFS does not perform well enough to use reliably over low speed WAN links, so practically speaking, a building-wide namespace is the best we can do.
- Network Intensive
  NFS caching is minimal, in core memory, and has no consistency. Heavy use of filesystems over the network results in heavy use of the network. The above limitations had wide ranging implications on how the pre-Aurora environment was built, such as how many server were necessary, how many copies of data were required, etc.

Design Goals

The features we wanted to have in the Aurora network filesystem included the following:

- Better Redundancy and Automated Replication
- Global access to shared files
- More efficient use of the network
- Better Security mechanisms

File System Selection

There are not too many options to choose from to meet the above requirements. NFS over TCP and CacheFS are solutions which do not meet all of our requirements and are not available on all the platforms we need to support. DFS would appear to be a possibly better choice, but the technology is not yet mature enough for use in a production environment.

AFS is currently the only production, supported distributed filesystem technology available meets a significant portion of our requirements. Some of the features of AFS which are key reasons why it was chosen are:

- Local Disk Cache
- Guaranteed Cache Consistency
- Logical Volume Management
- Automated Data Replication
- Transparently Available Redundant Data
- Superior Performance over WAN links

AFS is not the perfect solution for use as a Global Filesystem, however. There are a number of problems we have encountered, most of which we have been able to work around, and some have introduced new constraints on the design of the environment:

- One Global Cell
  The Ubik protocol used by the AFS database server does not scale well enough to allow us to have one global cell, covering more than 30 interconnected offices. This restricts us to having one cell per building.
- Inter-cell Data Distribution
  AFS provides a mechanism for replicating data within a cell, but there is no mechanism for distributing replicated data between different cells. We have had to develop a distribution mechanism from scratch internally.
- Kerberos Support
  We already used Kerberos on our systems. The version that we use, however, was different than the version required by AFS. As a result, we had to provide bridging mechanisms between the two systems.
- Sparse File Support
  There is none. Sparse files in non-replicated volumes work by accident, but sparse files in replicated volumes do not. This precludes use of AFS by a large class of internal data files, which will have to remain in NFS until we have true sparse file support in AFS, or eventually DFS.
- Byte-Range Locking
  File level locking is supported, but not byte-level range locking. This prevents many PC-based application from using AFS for user-data files, since use of byte-range locking is so prevalent in that environment.
- Backup Systems
  The AFS backup system is very weak, and third party support for AFS backups was non-existent. Early attempts to restore huge amounts of data have been very disappointing. We partnered with another vendor (Boxhill) and had them write a module (vosasm) that interfaces with Legato's Networker to provide volume-level backups. However the technology is still far from optimal.
- No Per File Permissions
  The file permission semantics change significantly between UFS and AFS. While

there are more options for directory permissions, there are no per file permissions.

- Significant Departure from UFS Semantics
  Error checking write() is not good enough anymore; you have to check close(). This change requires coding changes to bullet proof some applications, many of which we do not have control over (third party applications). The behavior of mmap()ed file is also significantly changed, so migration of data and user applications to AFS is not trivial in some cases.

Global Virtual Cell

The goal of a single, globally consistent view of a shared filesystem is still possible, even within the limitations of the technology provided by AFS. Although the granularity of individual cells is at the building level, this is hidden in the user view of the filesystem. Users will almost always access data through our "canonical name space", not being aware that some files come from the local cell, while others come from foreign cells.

Inter-cell access is provided by mount points for each cell, collected under /ms/.global; see Figure 1. The two character names refer to our locations, for example tk is Tokyo, ln is London, etc.

The pathnames used by users, applications, etc, reference a canonical pathname space. The top level view is shown in Figure 2. The four directories dev, dist, group and user make up the canonical namespace. A user home directory, e.g., /ms/user/w/wpm, is just a pointer into the global namespace to the actual location of the volume for that user; see Figure 3. Thus, wpm's home directory becomes transparently relocatable from cell to cell, without requiring changes to the canonical pathname to his home directory.

The same approach has been taken for the "dev" (Development Volumes, e.g., source code) and "group" (Shared Group Volumes), and together these 3 classes of data cover all the non-replicated RW data we maintain in AFS.

The final class of data, distributed replicated data, is available under /ms/dist, which consists of mount points for volumes assumed to be obtained from the local cell. Our internally developed volume distribution system automates the task of actually replicating the data between cells.

Real-time/Batch Auditing

There are no built-in mechanisms for auditing the state of AFS fileservers. In order to support AFS in a production environment we have had to spend a significant amount of time developing software to audit the state of AFS, both in real-time (monitoring AFS server process error logs) and in batch.

```
% fs lsm /ms/.global/*
'/ms/.global/bk.a' is a mount point for volume '#a.bk.ms.com:ms.cell'
'/ms/.global/ln.a' is a mount point for volume '#a.ln.ms.com:ms.cell'
'/ms/.global/ex.a' is a mount point for volume '#a.ex.ms.com:ms.cell'
'/ms/.global/mg.a' is a mount point for volume '#a.mg.ms.com:ms.cell'
'/ms/.global/tk.a' is a mount point for volume '#a.tk.ms.com:ms.cell'
'/ms/.global/sa.a' is a mount point for volume '#a.sa.ms.com:ms.cell'
```

Figure 1: Mount points

```
% ls -al /ms
total 17
drwxr-x 2     afsadmin 2048 Dec 19 1994  .
drwxr-xr-x 16 root      512 Jul 18 03:08 ..
drwxr-xr-x 2  afsadmin 2048 May 13 00:28 .global
drwxr-xr-x 2  afsadmin 2048 Jan  5 1995  .local
drwxr-xr-x 68 afsadmin 4096 Jul 17 13:06 dev
drwxr-xr-x 2  afsadmin 2048 Jul 10 14:31 dist
drwxr-xr-x 4  afsadmin 2048 Jun 22 18:02 group
drwxr-xr-x 29 afsadmin 2048 Jul  5 10:53 user
```

Figure 2: Top level view

```
% ls -al /ms/user/w/wpm
lrwxr-xr-x 1 afsadmin 27 Feb 10 10:56 /ms/user/w/wpm ->
                                      /ms/.global/sa.a/user/w/wpm
```

Figure 3: Global namespace pointer

Volume Management System (VMS)

In a multi cell environment such as ours, we are required to replicate data between cells. We developed a system to automate the distribution of data, using incremental vos dump/restore technology, and a configuration database to maintain timestamp information and master/slave volume relationships.

Canonical vs. Distributed Volumes

We have introduced the concept of a "canonical" volume, which is the master source volume for the "distributed" copies maintained in each cell globally. Changes are made to the canonical volume, and then incrementally dump/restored to the distributed volumes in each cell. The distribution mechanism works by dumping the backup copy of the canonical volume to a file, and forking multiple "vos restore" commands in parallel to all the cells.

Incremental Propagation

Incremental propagation is accomplished by managing the timestamps in a database external to the filesystem. The act of updating backup volumes or moving distributed volumes from server to server will invalidate the Last Update timestamps on the volumes themselves. Thus, when a volume is released for distribution the timestamp from the backup copy of the canonical volume is updated in the database.

Once the distributed volume in a given cell has been successfully updated, this same timestamp is updated in a separate table in the database for this volume in the given cell. Upon subsequent releases of a given volume, the timestamp for each separate distributed volume is the time from which an incremental dump of the canonical must be performed to bring it up to date.

Authenticated Access to Privileged Commands

In our pre-Aurora NFS-based environment, it was easy to delegate permission to distribute and maintain a portion of our distribution tree to non-root users, as rdist requires no special root privileges to distribute files from to server to server. AFS requires special privileges to dump/restore volumes.

VMS uses Kerberos mutual authentication to determine the identity of the user, and then performs various restricted operations on the user's behalf, using the user identity as the key to lookup authorized operations. This mechanism allows development groups to create new AFS volumes for development of a new release of an application without the necessity of system administrator intervention.

Our goal is to automate all the normal operational procedures for AFS which require special administrative privileges. Most of the processes which are automated have multiple steps and are very error prone when performed by human beings, even experts. VMS automates these steps, reducing operator error, and eliminating the need for giving junior administrators special privileges.

Growing Pains

Although the implementation of AFS for our Global File System is well on its way to success, it has not been without its problems.

WAN Issues

Filesystems across WAN links have historically been forbidden because of the ease of inducing heavy use of the WAN and the poor behavior of the filesystem technology (NFS). Use of AFS over the WAN is significantly better, given the behavior of the RX protocol under high retransmission scenarios. But 10MB of data over a 64KB link is still a heavy load regardless of the efficiency of the transfer mechanism.

We have to be very careful to ensure that access of non-replicated globally available data is minimized, since we have market data critical to our traders flowing over the WAN. Use of technology such as Cisco's custom queueing helps to minimize the impact of the problem, but a loaded WAN link still needs to be avoided if possible. This will prove to be a challenge as production usage of non-replicated data increases. Tools for analyzing WAN access and pinpointing the culprits during a saturation condition have not yet been deployed.

Training and Support

AFS is a radical new technology from the point of view of support personnel. Learning to manage the filesystem, debug and analyze problems, etc. requires understanding a new set of technology and tools. Training has been and continues to be a challenge, as we struggle to get existing administrators up to speed on both the vendor provided technology and the system we have developed internally to implement the global filesystem.

Current Status of the Global Filesystem

We currently have 26 buildings (i.e., potential cells) globally with UNIX hosts of some kind, and by the end of 1996 will have AFS available in all of them. The size of these installations ranges from 3 or 4 hosts to over 2000, and thus the server infrastructure varies from site to site.

In the larger cells, we have installed dedicated AFS file and database servers. In medium sized cells, we have dedicated servers performing both file and database server functionality. In smaller sites, existing servers are simply having additional disk space install for AFS service, and these servers will share functionality with other CPU and database functions.

We currently have approximately 20 GB of replicated data in AFS, and over 200GB of read/write user data (i.e., source code, user home directories, and group directories). We expect the amount of replicated data to grow steadily, but non-replicated data growth could potentially be explosive, with as much as a Terabyte on line by the end of the year.

Distributed Systems Database (DSDB)

Design Goals

As anyone who has attempted to manage a large set of systems centrally knows, there is an absolute need for a central repository of configuration information. This includes the traditional user and host information, but also extends to machine information, configuration files, software information and a legion of other information. In addition, the configuration database must provide significant dependency checking, so that, for example, a user cannot be deleted until all groups and mail groups that he owns are gone, any projects that he has responsibility for are reassigned, etc. The Global Configuration Database is part of virtually every aspect of the system.

Before attempting to build a system from scratch we surveyed the marketplace for possible solutions. Unfortunately, most products address environments in which there is only minimal central management, with many departments wanting their own autonomy. As mentioned earlier, we solved this problem using political, rather than technical means. We felt that allowing multiple administrative domains introduces the possibility, even likelihood, of local configuration changes. Our goal was to maximize homogeneity and minimize local changes. These products are designed to optimize for local customization.

We also felt that these products focus on allowing distributed autonomy meant that the functionality to allow total central administration suffered. Additionally, we wanted whatever system we used to tie in seamlessly to the other databases that were scattered around the company, such as the Human Resources and Inventory databases. Since we had already built databases like this in our pre-Aurora systems, we decided that we had the experience and knowledge to do a better job of building this database ourselves.

The goal of the system that we built, the Distributed Systems Database (DSDB) was to create a configuration database so that, in the event that we lost every building at Morgan Stanley, we could rebuild the entire physical environment with a backup of the DSDB and a lot of money. Note that this does not mean user data, only system configuration information. All system configuration information is primarily stored in DSDB, and only derived on the system itself.

Additionally, the database that was designed is a configuration database only. There is no real-time access component of it. For example, the source for the NIS maps are maintained in DSDB, but there is a separate process that dumps the information from DSDB to NIS, and NIS handles the real-time lookup queries.

In the pre-Aurora environment, we had two databases of information that contained information in the NIS maps and machine configuration information. However as we grew, we discovered some major problems with them:

- They were two separate, non-interacting systems
- There was very little consistency checking. The checking that was done was typically only on the input side. When someone deleted a user, there was no checks to see if that user was in other groups, mailgroups, etc. As a result, there was a lot of old cruft lying around.
- Because of the size of our maps, changes took over an hour to dump from the database and propagate worldwide.
- They were not designed to scale as much we were scaling our environment.
- One of the databases was based on a lightweight, locally developed database engine that was very flexible, but in which complex queries took a long time

DSDB was designed to replace both these databases and provide significant added functionality. DSDB was designed to:

- Provide extremely strong consistency checking
  You cannot add objects that do not fall into pre-specified criteria (for instance, you cannot add a host entry unless the network already exists) and you cannot delete an object that is referenced by anything else (you cannot delete a host for which there is an fstab entry in the database).
- Interact with other internal databases
  Other organizations had databases that were primary sources of information of information, such as the Human Resources database and the Inventory database
- Ensure information exists in only one place
  When a user changes their name, we no longer have to update it in dozens of places. They submit the appropriate form to the human resources department, and a short time later, the Gecos Field in the password file reflects the change.
- Complete historical information
  Information in the database is never deleted, it is aged. Using this, we can
  - get a snapshot of our environment at any given point in time

  - trace who did what to the environment
- Provide incremental propagation
  We can request all changes from the database from a given time. This allowed us to implement incremental propagation, which means that we now propagate all NIS changes worldwide within minutes, instead of days.
- All data is world readable (internally)
  There is no private data in the database. Based on the nature of the data that we were storing, we felt that this allowed us to easily avoid complex security issues.

In additional to the standard NIS information, we keep virtually all configuration information in this database. Some examples:

- fstab file information
- processes to start when a machine is rebooted
- machine configuration information, such as the devices on the system, the names of the mounted file system, etc

Primary vs. Secondary Information

There are two types of information kept in DSDB, primary and secondary. Primary information is that information for which DSDB is the primary source, such as the data in the password and host files. Secondary information is that information for which DSDB is simply a repository for ease of searching and reporting, such as the amount of memory on a machine, or the type of the graphic card, or even the user's real name (for which the human resources database is the primary source).

Primary information is typically entered manually as the user or machine or service or whatever is added. Secondary information is usually loaded in an automated, batch format. There are nightly audits that are run to collect information from all machines and upload this information to the database. This information allows us to create reports of system characterizations and usage.

There are a number of programs that run to get information out of the database. One such program is the NIS updater, which distributes the NIS maps incrementally. Another is the program to update the fstab file on a system after it has rebooted.

Architecture

Because the majority of our pre-existing internal databases are based on Sybase, we based DSDB on Sybase as well. All updates to the information are done through stored procedures. These procedures are responsible for the strong consistency checking.

The history mechanism that we use provides us with a means to do incremental propagation. For the NIS maps, every NIS server is a master. When it retrieves information from the database, it stores a timestamp in the YP_LAST_MODIFIED key of the map. Next time it queries the database, it asks for all changes since that timestamp. The sybase procedures to do this are tuned to make this a very fast operation.

Operating System Configuration

Design Goals

The primary goal of the OS Configuration portion of Aurora is to design an Operating System configuration which allows us to manage easily tens of thousands of hosts spread around the globe. The OS configuration is only concerned with system data, not with the data associated with, for example, user home directories, exported NFS partitions, or Sybase data partitions. We are addressing the files that are required to run the core system – those file under root, /usr, /var, etc.

Homogenous Support for Heterogeneous Hardware

We want to maximize the uniformity of the machine configurations to simplify administration, however we also require support for multiple hardware platforms, simply as a means of leveraging the right hardware for the right job to meet the varying needs of a wide variety of applications. From a hardware and operating system point of view, the system must support heterogeneous hardware platforms, however the operating environment must be as uniform as possible across all architectures.

Homogeneity is not easy to accomplish. Doing so in a heterogeneous environment is extremely hard. However we have shown in our previous environments that creating an environment like this reaps large benefits, including economies of scale and the ability to allow users to use the best tool for the job.

Rapid, Automated Installation and Reconfiguration

Hosts should be easily and rapidly reconfigurable. An installation mechanism which takes two hours to dump data from a CDROM, and then requires an administrator to update several locally maintained files, either from a backup system or from memory, is simply not practical when installing hundreds of machines. Furthermore, if a trader's workstation dies, minutes make a real difference. Our users require us to replace user desktops in under 10 minutes. Server OS reconfiguration should be just as fast, not accounting for possible restoration of data such as Sybase partitions.

Installation of systems are often done by people without the root password, such as electricians. When a machine is removed from the vendor's

shipping material, the installer should be able to plug it in, turn it on and wait till he gets the login prompt. The only work a system administrator should do is define the ethernet address in the host configuration database.

Control-Alt-Delete

When there is a problem with a user's desktop, it is not always appropriate to take the time to discover what the problem is. Often, time is of the essence, and the user just wants to get back on-line as quickly as possible. We allow users to do the Unix equivalent of "Control-Alt-Delete". If there is a configuration or disk problem, the user on a Sun, for example, simply types "L1-A", and then "boot net". The machine is completely scrubbed, reinstalled and rebooted in under 10 minutes, with no intervention by an administrator and no root access required. In fact, in Aurora it takes less time to reinstall a system than it does to fsck it's root partition.

In actual practice we discourage use of this feature by end users. The system administrator needs to make the decision that the problem requires a reboot, and then about whether discovering the root cause of the problem is more important than getting the machine back up again quickly.

Support for Machine Models

All customization of machine configuration will be handled via the configuration database, by associating the customizations for a particular type of machine with a "model".

For example, a "desktop" model would be used for most user workstations with graphical displays attached to them. However, the model for a generic headless CPU server in a comm room "server", may differ only sightly from "desktop". The former will support use of mirrored or striped filesystems perhaps, something we don't currently support on the desktop.

A "sybase" model may differ from the generic "server" model by configuring special directories in /var, installing a special version of a kernel (or configuring a special loadable kernel module for use).

An "afsfileserver" model will have a much larger number of locally copied replicated files in order to make the machine as independent as possible on the primary service it is providing.

Upgrade to new technologies

We used SunOS 4.1.3 and AIX 3.2.5 in the pre-Aurora environments, and have taken these operating systems to the limit in many ways. The vendors have begun putting more effort into the new operating systems, and the most modern hardware is only supported on the most recent versions of the operating systems. We chose to implement the OS configuration portion of Aurora entirely on Solaris 2.x and AIX 4.x in order to take advantage of the advanced available in these OS release.

Building from Scratch

We could have saved ourself some time by making use of the binary compatibility modes available in upgrading the operating systems. We made a conscious choice not to do this. Rather, every single program, binary and script was carefully examined, and its existence in Aurora had to be justified. This allowed us to bring forward only those applications, scripts, and methodologies which are required in the new environment, leaving behind a lot of historical infrastructure which is no longer necessary or simply outdated.

Dataless AFS Client Design

Many of the above design goals are met by implementing a dataless AFS client. In this design, local copies of replicated files (e.g., /usr/vice/etc/afsd) are present only in order to bring up afsd, from which point on, everything is accessed from AFS. Every file kept local to the machine has to justify its existence based on this criteria.

Most of the traditional pathnames for entire directories and most configuration files are merely symlinks into AFS. Prior to starting afsd, /ms is a directory which contains a symlinks to /localfs, where the real copies of locally maintained files reside. Once afsd is started, the AFS namespace overlays this, and files are no longer accessed from /localfs.

Minimizing the contents of /localfs is one of the keys to minimizing the installation and reconfiguration time of a host. The amount of data on the current Solaris 2.4 model is less than 20MB. At boot time, all local files are checked against "correct" copies of the file in a central AFS repository. If files are different, new files are copied to the local disk, and a reboot occurs. This allows us to always keep local files up to date.

One of the problems we had to solve with the AFS model was AFS cache initialization time. We use a fixed number of files for afsd start-up, and the creation of 400+ files in a large cache normally takes as much as 30 minutes to complete. This time is reduced to less than a minute by enabling asynchronous I/O during the afsd start-up.

Summary

As of the writing, the environment has reached the following milestones:

AFS

We have rolled out AFS to both the old and new environments. AFS is currently available on

about 50% of our machines worldwide. By 12/95, AFS will be available on all systems.

Global Desk

The implementation of Global Desk is complete. Currently, Global Desk is rolled out to 25% of our systems. We expect 100% installation by Q196. The rollout of Global Desk is slowed because each user must be changed and converted to the profile system.

DSDB

DSDB is running for Aurora and in parallel on our old systems, supplying all NIS information. In will be rolled out to all systems by 12/95. New functionality is being continuously added.

OS Configuration

There are currently about 50 "pure" Aurora desktop systems, based on Solaris 2.X. Work to support AIX 4.X is underway. Additionally, we have about a dozen production servers running under Aurora. We expect desktop rollout of the Aurora OS Configuration by Q196.

Acknowledgments

The following people were involved in the initial design of the Aurora System: David Birnbaum, Marc Donner, Chris Edmonds, Xev Gittler, Douglas P. Kingston, W. Phillip Moore, David Nochlin and J. Rambhaskar. Richard Campbell, Bruce Howard and Mike Lewis were also involved in the implimentation. Finally, thanks to Ben Fried, David Nochlin, Paul O'Donnell and Rebecca Schore who all proofread the paper and made valuable comments.

Author Information

Xev Gittler has been a Unix Systems Administrator/Architect for over 10 years. He received his B.A. in Medieval and Renaissance Studies in 1987. He started administering systems in universities, moved to R&D labs and finally to Wall Street, along the way managing systems from 3-5,000 machines. He co-founded and runs the New York System Administrators Group (NYSA), a local-area group affiliated with SAGE. He has no outside interests other than his wife and his child. He can be reached at xev@morgan.com.

W. Phillip Moore received a B.S. in both Physics and Math from the University of Oregon in 1986, and then entered the Ph.D. program in Physics at The Ohio State University, from where he escaped with an M.S. in 1988, after publicly confessing he was in reality an engineer. He then fled to Osaka, Japan where he spent 4 years as a UNIX/Network Systems Administrator for Matsushita Electric Works. In 1992, he joined Morgan Stanley Japan in Tokyo and worked in the Distributed Systems group, and became a founding member of the Core Infrastructure Group. Phil now works in the Morgan Stanley's New York office and heads the Global Filesystem Project. He can be reached at wpm@morgan.com.

J. Rambhaskar, received his B.E. degree in Mechanical Engineering from Shivaji University, India in 1990. During the Summer of 1991, he join the Systems Group, College of Engineering, Ohio University as a System Administrator. He joined Morgan Stanley as a System Administrator in October 1993. He later joined Morgan Stanley's Core Infrastructure Group in October 1994. Since then he has been involved in the OS Configuration and Global Filesystem projects. He can be reached at jram@morgan.com.

References

James Gettys, "Project Athena", pp. 72-77, *USENIX Conference Proceedings*, 1984.

John H. Howard, "On Overview of the Andrew File System", pp. 23-26, *USENIX Conference Proceedings*, 1988.

Michael Leon Kazar, "Synchronization and Caching Issues in the Andrew File System", pp. 27-36, *USENIX Conference Proceedings*, 1988.

Lessons Learned from Project Athena, Dan E. Geer, Jr. pp. 221-247, *Distributed Computing: Implementation and Management Strategy*, Edited by Raman Khanna, Published by PTR Prentice Hall, 1994.

John Leong, Project Andrew, in *Distributed Computing: Implementation and Management Strategy*, pp 203-220, Edited by Raman Khanna, PTR Prentice Hall, 1994.

Appendix – Details of OS Configuration Design

This section provides the details of how we designed the OS configuration for Solaris 2.X. In particular, it explains the installation process, the run time file system layout, how we update local system files, and how the start-up scripts work.

Installations

The Solaris 2.x Jumpstart installation procedure has grown more manageable since the SunOS 4.x suninstall. It provides pre-configuration information, etc. However Jumpstart does not provide complete automatic configuration. In addition, the overhead on the installation server is high, requiring 200+ MB of UFS disk space, and it does not support client builds across a router.

The design requirements for clients installation specified that the installation time should be under seven minutes, the effort to install a client should be the absolute minimum and the resources required on the installation server should be minimal. Our goal was to allow someone to simply

plug the hardware into a power source and the network, and turn it on.

The steps involved in the base Solaris2.x JumpStart (netinstall) are very simple.

1. Client send a rarp request and get the IP Information, then tftpboots inetboot
2. With the IP Information it then contacts the bootparams for root file system information, etc. This root file system entry in bootparams is KVM specific.
3. NFS mounts root and loads up the kernel.
4. The kernel remounts root by contacting bootparams again and starts up init

To achieve our design requirements, we did the following.

1. Fixed rarpd to dynamically assign IP information if no information is available for that host.
2. Fixed inetboot to autodetect KVM values so architecture information is not required in bootparams
3. Patched the NFS kernel module to detect KVM values so that it can do the right thing when it queries bootparams to remount root.
4. Truncate the /export/install (NFS install tree) to a very small distribution, approximately 20 MB per architecture
5. Separate out the installation server functionality into 2 parts
   a. local network functions services like rarp, tftp and bootparams
   b. NFS root filesystem.
6. Dropped pkgadd to install client packages as it was too time consuming. Instead, we created a prototype of the root partition and we use cpio to install it on the new filesystem.

Using this method, a single installation server can install an arbitrary number of clients. The only limitation is the bandwidth between the client and server, and the number of simultaneous installs. The actual installation process is as follows:

1. The machine is powered up, and a network boot is started
2. A rarp request is sent out to the network
3. The dynamic rarpd provides a hostname and ipaddr
4. The machine then tftps the inetboot kernel and runs the inetboot kernel
5. inetboot detects the KVM type and NFS

```
exmktaaa root=saaa2:/export/fid/bootnet/sparc.Solaris_2.4
sun4c=saaa2:/export/fid/bootnet/sparc.Solaris_2.4.sun4c
sun4m=saaa2:/export/fid/bootnet/sparc.Solaris_2.4.sun4m
wsmodel=localhost:aurora_install cellname=localhost:a.sa.ms.com
```

Figure 4: Typical bootparam entry

```
% ls -l /usr
lrwxrwxrwx 1 root other 19 May 24 11:47 /usr -> ./ms/dist/sunos.5.4
```

Figure 5: Example symlink during startup

```
% ls -al /ms
total 10
drwxr-xr-x 2 root other 2048 Jun 5 21:41 .
drwxrwxr-x 20 afsadmin sux 2048 Jul 27 15:53 ..
lrwxr-xr-x 1 root other 23 May 9 14:21 dist -> ../localfs/root/ms/dist
```

Figure 6: Contents of /ms before AFS is running

```
% ls -l /ms
total 34
drwxr-xr-x 2 afsadmin root 2048 Dec 19 1994 .
drwxr-xr-x 23 root root 1024 Jul 26 17:18 ..
drwxr-xr-x 2 afsadmin root 2048 May 13 00:28 .global
drwxr-xr-x 2 afsadmin root 2048 Jan 5 1995 .local
drwxr-xr-x 67 afsadmin root 4096 Jul 25 18:44 dev
drwxr-xr-x 2 afsadmin root 2048 Jul 28 14:30 dist
drwxr-xr-x 4 afsadmin root 2048 Jun 22 18:02 group
drwxr-xr-x 29 afsadmin root 2048 Jul 5 10:53 user
```

Figure 7: Contents of /ms after AFS is running

mounts the proper NFS root associated with the KVM type
6. The machine loads the /kernel/unix and all the required kernel modules and starts up init
7. init runs /sbin/rcS and /sbin/rc2
8. /sbin/rc2 starts up AFS and basic NIS services
9. After the basic Services are started up, the model script is run, which formats the boot disk (sd0), sets up the AFS cache and uses cpio to copy files to the local disk. The machine then reboots from the local disk.

Figure 4 shows a typical bootparam entry. "wsmodel" is the model name for a machine. "cellname" is the AFS cell name to use for installations. It need not be the local cell. "sun4m" and "sun4c" are the NFS roots depending on the KVM.

Boottime and Runtime Layout of the File System

The contents of the root file system on an Aurora workstation is minimal; it only contains files that are required to bring up AFS. The remainder of the contents of the root file system are symlinks back into /ms (which is our AFS mount point). Since /ms is not available before starting AFS, we have a UFS /ms which has symlinks pointing to /localfs. When AFS starts up, these symlinks are hidden from the system; see Figure 5 for an example. Figure 6 illustrates the contents of /ms before AFS is running. /localfs contains files that will be overlaided when AFS is started up. Figure 7 shows the contents of /ms after AFS is running.

Updating Local File Systems

A master copy of all files that are required on the local disk at boot time is kept in an accessible AFS directory. Every time a machine reboots, a script is run to check if there is any file in the repository that differ from the files on the local disk, if new files have been added, or if old files have been removed. In addition to the common repository, there is also a provision for overriding files on particular machines.

The update script takes about thirty seconds to run and makes sure that all files in UFS are identical to the master's. If any changes are applied to the UFS as a result of this script, the machine is rebooted. This allows us to update kernels and other locally required files and ensure that they are always up to date.

Start-up scripts (/etc/rc*)

Start up Scripts needed before AFS are maintained in the UFS. The rest of the start-up scripts are symlinked back into an AFS directory. For example, the SunOS default /sbin/rc2 executes scripts from /etc/rc2.d. In Aurora, /sbin/rc2 runs scripts from /etc/rc2.preafs.d and then from /etc/rc2.d. /etc/rc2.d is a symlink back into AFS. This means maintaining scripts across all machines becomes trivial.

Machine specific start scripts are started out using "start" which is a locally written script which reads from central file and evaluates whether a particular process should be started on a particular machine. By using this mechanism, we avoid the need for different rc scripts on different machines.

How to Upgrade 1500 Workstations on Saturday, and Still Have Time to Mow the Yard on Sunday

Michael E. Shaddock, Michael C. Mitchell, and Helen E. Harrison – SAS Institute Inc.

ABSTRACT

Imagine: It's Saturday afternoon. You run a script, watch it for a while, then go home. When you come back the next day, 1500 workstations and fileservers have new operating systems installed, complete with all your local customizations, with the user data on each one undisturbed and without leaving your office. On December 17, 1994, we did just that.

This paper will describe the infrastructure that allows us to perform completely automated updates of a large distributed network of HP UNIX computers. First, we will describe the policies we designed for distributed systems administration. Next, we will describe the tools which we developed or collected to implement these policies, and we will describe how to put it all together to do an upgrade. Throughout we will explain the philosophy behind it all and how our particular implementation could be generalized to other sites. Finally, we will describe some of the lessons learned along the way.

Support Philosophy and Design Goals

In order to support a large number of workstations and fileservers with a small number of system administrators, we decided very early in the design phase of our network to do everything possible to make all of the machines look the same, but still allow for per-host tailoring. This goal was helped considerably by the fact that our network consists of only Hewlett Packard 9000/700 series workstations and fileservers.

The second design goal was that we try not to modify any more system files than necessary. This would allow us to move from one operating system version to another without having to track down a large number of system files that we changed in each version.

Our basic philosophy is that network services should be centrally administered, and should be replicated and distributed. AFS replicated volumes and BIND, the Berkeley Internet Name Daemon, are excellent examples of how we wanted to do things. Each uses a master copy, which is then replicated to multiple distributed service providers. If any one service provider becomes unavailable, the service requesters would automatically shift to another service provider. We wanted both our day-to-day systems and our support infrastructure to follow this paradigm.

In addition to using replicated distributed services, we also try to cause each workstation to use the service provider which is "closest" to it on the network. Our network is heavily subnetted, and we want to reduce inter-subnet traffic and network latency.

System Design

Our standard system configuration is an HP700 with two internal disks. We named the two internal disks / and /local. The root disk, /, is virtually identical on every workstation and fileserver (except for licensed software differences which are managed by software distribution tools), and is where the operating system binaries, such as /bin and /usr/bin, reside. The /local disk is used for data that is machine specific. This machine specific data includes user home directories, backup tables, the workstation's AFS cache, and other local configuration files. As a result of this design we were able to set up a disk "cloning" system. If a workstation loses its system disk, we are able to replace it quickly and easily. If a /local disk breaks, we have everything that we need on the system disk to bring the system up to the point of being able to restore the necessary data on /local.

Since almost all of the machine specific data is located in /local, and since all of the system disks are virtually identical, the amount of data that we actually need to back up is greatly reduced. There are still a few files in / that need to be backed up, but this number is very small. If a system has more than the standard two internal disk drives, the additional drives are typically mounted as subdirectories of /local. Since our backup software traverses directory trees in a manner similar to *tar*(1), mounting a new disk under /local automatically adds it to our list of things to be backed up. HP-UX for HP 700 series machines does not support disk partitioning, so we were limited to using multiple disks in order to segregate system data from other data, specific to that machine. One could, however, achieve similar

results by using physical disk partitioning on other systems which do support it.

Tools

Hostclasses

Hostclasses are a way of using a symbolic name to define a set of machines, and to use set operations upon those sets. They were initially designed and implemented at MCNC [1]. This initial implementation read the hostclass information directly from files in a known location. We have modified hostclasses to use a client/server approach, which includes multiple replicated servers, in keeping with our overall philosophy. Hostclasses can be incorporated into applications either through a user level program or a set of C library functions.

Hostclasses can be used for myriad applications. For example, we have one hostclass called loc.DC. This defines all of our machines in our main Data Center. We also have a hostclass called appl.AFS, which lists all of our machines which are AFS fileservers. The intersection of these two hostclasses,

```
=loc.DC % =appl.AFS
```

lists all of the machines in our main Data Center which are AFS fileservers. One of our most common uses for hostclasses is to list which machines have which extra products installed, such as the ANSI C compiler or Japanese NLIO support. Hostclasses allow for centralized list management independent of any individual application. A hostclass is similar to a *netgroups* (4), except that hostclasses are designed to be used in a more general way.

Sasify

Our primary software configuration management tool is called `sasify` (formerly called `doit`). It uses a central database called an *action file*, which contains a list of actions to apply to a host, and applies them. These actions can include downloading and installing a new kernel, deleting files, installing patches, etc. There is a file stored on each machine that defines the current `sasify` *level.* When a system reboots, it performs its normal startup procedures, then downloads a copy of `sasify` from one of a set of known servers, and runs the downloaded `sasify`. `Sasify` checks to see what the current `sasify` level is, downloads a copy of the action file that pertains to the local host, and performs any actions necessary to update the system to a new level. There are also ways to specify actions that should always be run before and/or after any level-specific actions.

`Sasify` uses hostclasses to determine which actions at a specific `sasify` level are to be run on which hosts. For example, your first level-specific action might be "for all machines that do not run X.25, download version 9.77 of /hp-ux." Your second action would probably be "for all machines that run X.25, download version 9.77_x25 of /hp-ux." In the first instance, we would use the hostclass expression

```
=sasify.HP700 - =appl.X25
```

whereas in the second instance we would only need to specify

```
=appl.X25
```

Following our support paradigm, `sasify` keeps a single centralized "database" of all of the actions necessary for all of our HP 700 series machines. After a new version of the action file is installed, it is replicated to our `sasify` database servers. When `sasify` downloads a new copy of the action file, it actually gets it from one of the five database servers.

In addition to the action file being replicated from a central location, all of the data that `sasify` downloads is also replicated from a central location. `Sasify` then picks the "closest" of the data replicas. If it cannot reach the data replica that it prefers, it will randomly pick another of the replicas.

`Sasify` can be used to maintain multiple configurations. It uses hostclasses to determine which configuration a machine should use, so it is not limited to supporting just one type of machine architecture. When we were designing `sasify`, we knew that we would eventually want to use it to maintain systems other than our main HP network, and designed it accordingly. We currently support 4 distinct HP configuration models and are working on extending `sasify` database to include the Suns that we support as well.

Since `hostclasses` and `sasify` were presented at LISA VI in 1992 [2], we will not go into more detail about their internal workings.

There are a number of additional software maintenance and distribution solutions that have been developed at other sites. These include package [3], depot [4, 5], and config [6], each with a different feature set, which are designed to solve similar problems. Most of these other packages specialized in tracking local software updates. We designed our package so that it would not only enable us to download new software versions to our workstations, but would also provide methods for adding and deleting files, and for running arbitrary programs. If you have not already adopted a formal software distribution system there is a good chance that you will find one already written which will meet your needs.

Getticket

Experience has shown that even though we have replicated most of our services, there are still times when we want to control exactly how many systems are accessing a service simultaneously. In

addition there are occasionally some services which are inherently difficult to replicate, perhaps because of licensing restrictions, or which generally receive only incidental use, but may be accessed more frequently during an upgrade. For example, all of our extra HP products, such as the ANSI C compiler, are loaded from a single location and are accessed only when a system disk is replaced or a product is installed on a new machine. During a full upgrade, however, we reload this software on each machine which is licensed for it. To provide this controlled access to these services, we wrote `getticket` and `ticketd`. The client program, `getticket`, queries `ticketd` for a *ticket* to a particular service. `Ticketd` knows which hosts provide which service, and how many tickets are available for that service on each host. It then hands out tickets to this service in a round-robin fashion to distribute the load between the service providers. The ticket that is given out has the name of the service provider embedded in it, so any scripts that we write do not have to know anything about who the service providers are, or how many of them there are. `Getticket` is also used to return tickets to `ticketd` once they are no longer needed.

The `ticketd` program handles the tickets for multiple services simultaneously. Each service is defined in a `tickettab` file. The `tickettab` file lists the service name, the ticket lifetime, and the names of the service provider and a count for that provider. Each service provider can have a different number of `tickets` which it contributes to the pool of tickets for that service. The `tickettab` file for the `hpux_patches` service might look like:

```
service  hp-ux_patches 14400
milton   8
hasbro   4
tonka    16
```

This specifies that for the service `hp-ux_patches`, all tickets will timeout after 14400 seconds (four hours). There are a total of 28 tickets in the service pool: 8 from `milton`, 4 from `hasbro`, and 16 from `tonka`.

The `ticketd` program builds a ticket out of the service provider name, an underscore, and an internally generated number (the seek offset into a file). A ticket from the `hp-ux_patches` service might be "milton_080". The entire string is returned to the `ticketd` program so that the service provider section can be pulled out of the ticket with the Korn shell syntax "${ticket%_*}", assuming the ticket is in the variable "$ticket" (and the hostnames do not contain underscores). The command

```
echo $ticket | awk -F_ '{ print $1 }'
```

is another way to get the service provider part of the ticket.

While `getticket` was designed originally to manage access to services on specific hosts, it can be used more generally. The service provider field can be any string. It does not have to be a hostname. This feature allows the `getticket` program to be used as a simple licensing agent. For instance, suppose you have a 20 user license for the image viewer `xv`. The `tickettab` file might looks like this:

```
service xv 1209600
xv 20
```

You could then use a simple wrapper program that uses the `getticket` library routines to get an `xv` ticket, runs the real `xv` program which has been hidden away, then returns the ticket. This is a handy way of complying with licensing restrictions on programs which do not support license management.

Netdistd, Update, and Filesetload

HP provides two programs with HP-UX that are used for distributing software across a network. The first of these, `netdistd`, is the distribution server. A `netdist` area includes software subsets, which HP refers to as *filesets*, and patches. The other program is `update`, which communicates with a `netdistd` to download filesets to the local machine. `Update` connects either to a single default `netdist` server or an alternate server specified on the command line. In order to make HP's update system fit our support paradigm we wrote a wrapper program called `filesetload`. `Filesetload` is told which service to use, and which filesets to load. `Filesetload` then checks to see if any of the specified filesets need to be installed, and if so, it uses `getticket` to get a ticket to the specified service, runs `update` to download and install the filesets, uses `getticket` to return the ticket, and then sends the log from the update to a mailing list of system administrators. Since `filesetload` is run every time a machine reboots, an unexpected benefit is that whenever a system disk is replaced any additional licensed products will be automatically reinstalled.

Sortaddrs

During an operating system upgrade, there are many machines rebooting simultaneously, each downloading a large amount of data. To prevent network bottlenecks it is helpful to balance the network load. `Sortaddrs` was written to address this problem. `Sortaddrs` takes a list of hostnames, sorts them by subnet address, and prints out the sorted list. The list is sorted so that the first entry is from subnet A, the second is from subnet B, and so on, until we cycle back to subnet A.

Putting It All Together

HP-UX Recovery System

HP, like most UNIX vendors, provides tools to build a memory-resident operating system and filesystem. They refer to this as a *recovery system*. The typical use of a recovery system is to create a tape to be booted when a system suffers from catastrophic failure of its system drives. We have used these tools to build a custom version of the recovery system with our support tools installed, which is only used during an operating system upgrade. During an upgrade of this type `sasify` installs some support files in /local, downloads the recovery system as `/hp-ux`, and then reboots. When the system boots, it is then running our recovery system, which does not access the internal disks. We then mount these disks under temporary names, copy any "precious" data from / to /local, unmount /, `newfs` the / disk, download a *dump*(8) image of the standard system disk, and restore it as /. Next we copy the previously saved precious data from /local back to /, and reboot again. At this point, `sasify` picks up where it left off, and continues with any remaining updates.

Since the recovery system has a limited size, we had to write smaller versions of some of the standard system utilities. For example, *mount*(8) takes up 180 KB of disk space. Our "expert friendly" version of *mount*(8), which does no significant error checking, but which is sufficient for our use while running the recovery system, only uses 12 KB. We were able to realize similar savings with several other programs that we needed on our recovery system.

Since the HP recovery system uses a memory resident file system, all of the programs necessary for the recovery operation (`shell, mount, restore, cpio,...`) are contained in the data segment of the kernel image. We had to choose very carefully what would go into the recovery image, because the HP boot ROM would not load a kernel bigger than about 6 MB. We also wanted the recovery image to be as useful as possible, so we included a few things we could have copied to the /local disk instead of leaving memory resident. We

```
  1 /etc/disktab          table of disk-drive geometries, used by 'newfs'
131 /etc/fsck             make sure the filesystems are OK
164 /etc/init             runs the /etc/rc actions
  1 /etc/inittab          tells /etc/init what to do
 16 /etc/newfs            initializes the filesystem
119 /etc/mkboot           installs the bootstrap program
 57 /etc/mkfs             makes a new filesystem, called by 'newfs'
 20 /etc/mknod            makes a 'special' device
 12 /etc/mount            mounts a file system
  1 /etc/rc               startup script
 12 /etc/reboot           reboots the system
 65 /etc/restore          restores a dump image
 20 /etc/umount           unmounts a file system
 25 /bin/chmod            change the protection modes of a file
 20 /bin/chown            change the owner of a file
 49 /bin/cpio             file archiver
 16 /bin/date             set/display the time
 90 /bin/dd               changes blocking factor of data
 98 /bin/gzip             compress/decompress program
 16 /bin/ln               links two files together
172 /bin/ls               get a directory listing
 86 /bin/mkdir            make a directory
 94 /bin/rm               removes a file
 32 /bin/sed              stream editor
262 /bin/sh               command interpreter
 29 /bin/stty             set terminal characteristics
 12 /bin/sync             updates super-block
 70 /lib/dld.sl           dynamic loader
856 /lib/libc.sl          shared C library
317 /usr/lib/uxboot.700.gz compressed bootstrap program, used by 'mkboot'
```

Figure 1: Included Programs

pared down the size of executables wherever possible, by writing our own simplified versions of `reboot` and `mount`, by using `gzip` to compress the boot strap loader (`uxbootlf.700`) installed on the system disk by `mkboot`, and by stripping the symbol table off anything that had a symbol table. Figure 1 shows a list of all the programs we included and their sizes, in KB. We could save some space by including the `/etc/unlink` program instead of `rm`, but we would also have to write our own `rmdir` program. The `ls` program is an extravagance; we would like to find something smaller, but `echo *` is not as easy to use. We would also like to include `tar`, but it weighed in at 200 KB. For our purposes, `cpio` at 49 KB does just as well.

The `/etc/rc` script used by the recovery system first tries to mount the /local disk. If that succeeds, it then executes a shell script placed on the /local disk (/local/recover/recover) by sasify. If the mount fails or the shell script is not found, it prints a message on the console and the shell is started. This lets us use the same recovery system for system updates and for emergencies.

We use `sasify` to load into the /local/recover directory any programs needed to finish the update. Currently that includes `find`, `hostname`, `ifconfig`, `sum`, `telnet`, and a few others. `Find` is used to generate a list of filenames to save before the update. `ifconfig` and `hostname` are used to set up the networking so that the dump image and check sum file can be pulled across the network. `Sum` is used to verify that no errors occurred in the transfer of the dump image. `telnet` is used to send a last-gasp error message when a failure occurs from which we cannot recover. We use `telnet` to connect to the SMTP port of our mail gateway and send it a hand-crafted SMTP message.

How Many, How Quickly

As previously mentioned, during our testing phase we can determine how long the update process takes. We also time how long certain phases of the update process take. We can use these timings, along with our knowledge of how many service providers we have and how many simultaneous connections the service providers can support, to calculate how quickly we can reboot the machines. We also know the total time we want the update to take. If our reboot rate cannot get all of the machines updated in the time frame that we want, then we know we will need to adjust the number of service providers where possible. For example, lets say that a workstation takes 30 minutes for a complete update, we do not want any more than 5 machines talking to a single update server at the same time, and we have 10 update servers. Since we have 1500 workstations, that means it will take at least (1500 * 30) / (5 * 10) minutes or 15 hours to complete the update. The workstations should be rebooted (15 * 60 * 60) / 1500 or 36 seconds apart. This gives an upper bound for the reboot interval.

In most circumstances we can reboot the machines much faster. Only a portion of the 30 minutes it takes to update the machine needs to be rate-limited to 50 machines at once. A fair amount of time is taken by checking disk consistency, mounting disks, enabling the network, and other local processing. The only part that has to be rate-limited is the section that downloads data from the `sasify` and `netdist` servers. If the average machine spends only 15 minutes of the 30 minutes downloading data, we can reboot a machine every 18 seconds instead of only every 36 seconds. The entire update would take about 8 hours instead of 15 hours.

There is a danger in updating machines this quickly. Since it takes 30 minutes for one machine to finish, and machines are updating every 18 seconds, if there is a mistake in the update procedure (30 * 60) / 18 or 100 machines could be affected before anyone notices the failure of the first workstation.

Doing the Upgrade

Since all our workstations run `sasify` as part of their normal boot-up processing and since `sasify` is doing the updating, all we have to do to trigger an update is reboot a machine. To facilitate rebooting 1500 machines we wrote a Korn shell script that reads a list of hostnames and a time delay value. It reboots each machine in the list and waits the specified number of seconds before rebooting the next machine in the list.

The first version of the script used a passed-in value as the time delay parameter. We soon realized that we needed a way to change the delay parameter while the updates were in progress. If you miscalculate how quickly to reboot machines the servers could become overloaded with update requests. In the calculations above we guessed that a server could handle five connections simultaneously. What happens if the servers can only handle four connections? We wanted a knob we could turn to speed up or slow down the reboots.

We added this knob by having the reboot script read a file containing the delay time every time it was going to delay. We now pass in a file name instead of a delay time. When we start upgrading the workstations we try to use a larger delay than is really necessary. We then watch the server load as the workstations start updating. As the workstations complete their upgrade successfully we decrease the time delay in steps, thus rebooting machines more quickly, until the calculated frequency is reached.

Upgrading Servers

One tricky problem in an automated procedure of this type is upgrading your upgrade servers. Special care is needed while upgrading these servers,

since some of them also run the various servers needed for doing the upgrades. We have managed to segregate the various servers well enough that we now do our global reboots in waves. First, we reboot our main `netdist` server. Then, we reboot the "database" servers one at a time. The database servers run `named`, the various AFS database processes, the hostclass servers, and the librarian services for `sasify`. Then we reboot half of the replica servers. These servers act as AFS replica servers, as `netdist` servers for HP patches, and as the download point for `sasify` data. Next we reboot the other half of the replica servers. At this point, all remaining workstations and fileservers can be rebooted.

Lessons Learned

Replicate, Replicate, Replicate

We discovered that it is important to replicate as many of your services as possible. This improves reliability, provides for load balancing, and allows for improved throughput for large-scale operations such as software updates.

Cleanly Segregate Functions

Try to segregate different functions on appropriate servers as much as possible. At one point, we were running AFS database servers on one set of machines, `named` on another set, and `sasify` librarian and data servers on a third set. In this case, trying to determine the reboot order was a nightmare. We eventually realized that named, the AFS database servers, and the librarian services all depended on each other. We relocated them onto the same set of machines, reducing the complexity of the problem significantly. Additionally, once we instituted the AFS replica servers, the reboot sequence became obvious.

Updating an Update Server

After a reboot, an average system will start its standard processes, run `sasify`, and then start its individual local processes. This means that on the `netdist` server machines we need to start any `netdist` servers before we start `sasify`. In addition, we modified `ticketd` and `sasify` to be smart about picking servers. If machine A asks for a ticket to a service that machine A provides, `ticketd` returns a ticket for machine A instead of whatever ticket was next available in the round robin queue. We made similar changes to `sasify`.

Centrally Administer Replicated Services

Central administration of our replicated services has made it much easier for us to maintain all of the necessary configurations. While this is not a luxury that some sites, particularly universities, have, we should still acknowledge its benefits.

Do Less, More Often

We discovered that it is easier and less risky to apply a few changes once a month than to apply a large number of changes a few times a year. This should be self-evident, but it did take us some time to realize this. This also allows us to track patches from HP more closely than we previously could.

Since we run an operation that is expected to be available 24 hours a day, 7 days a week, our upper-level management had to be convinced each time we need to do an upgrade that, overall, it was worth the downtime. Scheduled downtime only occurred 2-3 times per year, with many changes occurring at each of these outages. We were able to convince our management that the chances of a major error being made while only making a few changes was much smaller than if many changes were being made. We now schedule our downtime on a highly predictable monthly schedule with specific dates announced months in advance, This allows our product developers to schedule releases, regression tests, etc., without being surprised by scheduled downtime. In addition, we consolidate hardware changes to coincide with these scheduled maintenance times which has reduced the need for incidental downtime at other times. We are currently updating 1800 workstations each month.

Testing

When upgrading over 1500 workstations, a widespread failure can be particularly catastrophic and take a long time to fix. In this environment carefully controlled testing is extremely important before a major upgrade. We have a set of test machines where we do our initial testing. Once we are satisfied with these tests, we install the changes on the workstations of the UNIX support group. This gives us a chance to "test drive" the changes. During these tests we also time how long it typically takes to do an update, and use those numbers to help us determine how quickly machines can be rebooted.

The Time It Almost Didn't Work

When we first started updating the servers we had some network hardware problems. The symptoms were that the `sasify` program would hang without completing the transfer of the system disk image. We were worried that more network hardware failures could cause many workstation updates to fail in such a way that we would have to walk to them and restart them. We quickly wrote a program that would `fork()` and `exec()` its argument list and wait a specified number of seconds before killing its child process. This program was used to limit the time `sasify` could be hung. If `sasify` failed our script would restart it at the beginning (up to ten times). This change to fix the hung session problem was put into place after some testing.

We started the workstations while we analyzed the failures from the hung machines, still trying to find the original problem even though we had worked around it. Analysis of the hung machines suggested some additional changes could help prevent some of the failures. The new changes were added while the workstation update was in progress. There was little or no testing done to the new changes. (Big Mistake!)

Half an hour latter we started noticing that workstations were not finishing their update. We started to look for a reason and found a syntax error in the new changes. About 100 machines were trying to load the wrong things. We stopped the update process for the rest of the machines, then tried to figure out how to fix the 100 'broken' machines. We came up with a solution that required us to just restart `sasify`, tested it on a few machines, then started to walk to the 100 'broken' machines. Our use of `sortaddr` insured that they would be spread out all over our campus.

When we got to the first machine, we found it had already restarted `sasify`. We watched it until we knew it was running correctly, then we moved on to the next machine. It too was re-running `sasify`. It was then that we realised that the code to limit the execution time of `sasify` had kicked in, causing each machine to start over, this time executing the corrected code. It saved us from having to walk to 100 machines!

Conclusions

While the implementation that we describe was done on an HP platform, we believe that many of the concepts are generalizable to any UNIX platform. The specific tools that we use to manage this network are only part of the whole picture. You also need an overall policy which will guide your support structure development as your network grows. We have found that a comprehensive strategy, consistently applied across all support solutions, not just those designed specifically for software updates, makes performing major software updates highly efficient, even in a large network.

Availability

For information on the availability of any of the tools mentioned in the paper, please send email to heh@unx.sas.com.

Author Information

Helen E. Harrison is the UNIX Support Manager at SAS Institute Inc., where her group provides hardware and software support for a network of over 1800 UNIX workstations and servers. She has been involved in UNIX systems administration for over 12 years and holds a B.S. in Computer Science from Duke University. Reach Helen at SAS Institute Inc, SAS Campus Drive, Cary, NC 27513; or by e-mail at heh@unx.sas.com.

Michael Mitchell is a Systems Programmer in the UNIX Support Group at SAS Institute Inc. He has been involved in Distributed Computing for over 8 years and UNIX systems for 15 years. He holds a B.S. in Computer Science and a B.S. in Electrical Engineering, both from North Carolina State University. Reach Mike at SAS Institute Inc., SAS Campus Drive, Cary, NC 27513; or by e-mail at mcm@unx.sas.com.

Michael Shaddock is a Systems Programmer in the UNIX Support Group at SAS Institute Inc. He has been involved in UNIX systems administration for over 8 years and holds a M.S. in Computer Science from the University of North Carolina at Chapel Hill. Reach Mike at SAS Institute Inc., SAS Campus Drive, Cary, NC 27513; or by e-mail at shaddock@unx.sas.com.

References

1. Helen E. Harrison, Stephen P. Schaefer, and Terry S. Yoo, "Rtools: Tools for Software Management in a Distributed Computing Environment," *Proceedings of the Summer USENIX Conference*, pp. 85-93, San Francisco, CA, June, 1988..
2. Mark Fletcher, "doit: A Network Software Management Tool," *Proceedings of the USENIX Systems Administration (LISA VI) Conference*, pp. 189-196, October 19-23, 1992., Long Beach, CA.
3. Transarc Corporation, "AFS System Administrator's Guide," FS-D200-00.10.4, pp. 14-1 - 14-26, Pittsburgh, PA.
4. Walter C. Wong, "Local Disk Depot - Customizing the Software Environment," *Proceedings of the USENIX Systems Administration (LISA VII) Conference*, pp. 51-55, Monterey, California, November 1-5, 1993.
5. Wallace Colyer and Walter Wong, "Depot: A Tool for Managing Software Environments," *Proceedings of the USENIX Systems Administration (LISA VI) Conference*, pp. 151-159, October 19-23, 1992., Long Beach, CA.
6. John P. Rouillard and Richard B. Martin, "Config: A Mechanism for Installing and Tracking System Configurations," *Proceedings of the USENIX Systems Administration (LISA VIII) Conference*, pp. 9-17, September 19-23, 1994., San Diego, CA.

Security Administration in an Open Networking Environment

Karen A. Casella – Sun Microsystems, Incorporated

ABSTRACT

As networking technologies evolve and business needs change, traditionally isolated and secure communication networks are giving way to more open computing environments. Security, network and systems administrators must therefore concern themselves not only with firewall and boundary security, but also with individual system security. Security administration in a large open network is a challenging assignment and requires a combination of auditing, assessment and compliance mechanisms. For very large networks, automation is another variable which is critical to consider in this equation. There are several tools available to assess the security of networks and systems; however, there are few freely available solutions for addressing the problems that these analysis tools detect.

This paper describes the changing network security paradigm and discusses what tools are available for identifying security vulnerabilities in an open network environment. It goes on to state the problem that we faced at Sun and describes the suite of tools that we have designed and implemented as a solution, focusing on the automation of system security assessment and compliance. Finally, *SunSWAT*, the Sun Security Weakness Attack Tool, is introduced and its evolution from a single shell script designed to respond to the results of a network security audit, into a system for improving system security, implementing enterprise security standards and auditing to those standards, is discussed.

Background

The traditional model for network security has been to fortify the boundaries of the network, monitor the perimeter and trust everything and every one inside. For many networks, boundary entry points are limited to a single Internet connection and perhaps a modem or modem pool. The Internet connection is typically made through a carefully controlled firewall complex. Modems are configured for dial-back or one time password schemes. Boundary security is tight and relatively easy to administer and monitor.

Picture a castle with a moat surrounding it with one or two heavily-fortified drawbridges. Everyone within the walls of the castle trusts each other; there are no locks on doors.

Emerging communications technologies, combined with changing business needs, have caused many companies to rethink this model of network security. Some examples follow:

- To keep pace with the competition, many companies are opening up their networks to business partners via the Internet or private network providers.
- Connecting small remote offices using dedicated circuits is not always cost effective, so some companies are investigating alternate networking solutions, including ISDN or the use of private or public networks for data communication.
- Telecommuting is a growing interest for many companies. Employees are beginning to work at home more to allow the company to save money, to keep employees happy and in some states, to comply with clean air regulations.
- Similarly, nomadic computing is taking off. Many employees who travel extensively, especially those in the sales force, carry their work environments on laptop systems, performing their day-to-day business from hotels, airports and phone booths.
- Many companies are finding themselves with a rising contractor workforce, which introduces new challenges for network security and systems administration groups.

These situations and others like them are forcing companies into a more "open" security model for their internal networks.

To support this new network paradigm, security must be maintained not only at the firewall and on critical server systems, but also on the individual desktop systems, i.e., every system on the network must be as secure as possible without negatively impacting usage.

Picture a modern day city. Nobody trusts anybody; there are locks on every door.

Herein lies the challenge. How can system support personnel effectively audit and maintain compliance to security standards (which often change) on a large, open network? There are many tools available to systems administrators to assess network and system security, but there has been little work done to assist with the automation of security compliance.

Network Security Tools

Tools that can be used to secure networks and systems generally fall into three basic categories.

Network security auditing tools act on a network of systems, scanning systems from outside of each system, for vulnerabilities that may allow unauthorized access to systems.

System security assessment tools are designed to examine individual system security from within the system by searching for conditions or configuration errors that may cause a system to be vulnerable to attack.

Security compliance tools are used to correct configurations and patch security holes on systems.

Several freely available tools exist that provide for system or network security auditing and assessment; however, the number of non-commercial solutions for actually correcting the security vulnerabilities that they discover is limited.

The following sections describe some of the tools that are available in each of these categories.

Network Security Auditing Tools

AutoHack [1] is a tool for auditing the security of a large TCP/IP network. *AutoHack* attempts to exploit well known vulnerabilities and generates reports based on what it finds. *AutoHack* is currently being used only within Sun Microsystems for network security auditing.

SATAN (System Administrator Tool for Analyzing Networks) [2] is similar to *AutoHack* in that it scans systems connected to the network noting the existence of often exploited security weaknesses. SATAN also investigates the "web of trust" in network security and offers a tutorial that explains what problems are discovered and what can be done to address them. SATAN is freely available in the public domain.

ISS (Internet Security Scanner) is also a multilevel security scanner. Since its initial release, ISS has become a commercial product and is no longer freely available.

System Security Assessment Tools

COPS (Computer Oracle and Password System) [3] examines a system for a number of known weaknesses and alerts the system administrator to them. COPS does not attempt to correct or exploit any of the potential problems it finds. There is an option to generate a file containing shell commands that attempt to fix a subset of the problems found, but it must be used very carefully.

The **tiger** [4] package of system monitoring scripts is similar to COPS in what it does, but is more extensive in the checks that it performs and is somewhat easier to configure and use.

Security Compliance Tools

The **secure_sun** script by David Safford checks and fixes fourteen common SunOS 4.x security holes. This script does not address Solaris 2.x systems or other operating systems and is limited in its scope with regards to the checks and fixes that it performs.

The Problem

A combination of the three types of tools described in the previous section is required to secure a large, open network.

To identify which security vulnerabilities exist, it is important to regularly audit security by centrally scanning the network and identifying which systems are vulnerable to external threats. Then, each of those systems must be examined to determine which conditions exist that allow for exploitation. Finally, the conditions must be corrected to adequately secure the system.

On a network the size of Sun's (nearly 30,000 hosts), this is no trivial task. Automation of the network security auditing, system security assessment and security compliance functions is critical.

Sun's Solution

Our approach for improving network and system security was to create a suite of tools for auditing, assessment and compliance. *AutoHack* was developed to perform the network security auditing and reporting. We may have been able to use one of the freely available system security assessment tools; however, we still needed a security compliance tool. Since none of the freely available tools matched our requirements very well, we decided to develop a system that combined the system security assessment and compliance functions.

Design Goals

The design and implementation of *AutoHack* are beyond the scope of this paper and are described in a paper by Alec Muffett for the 1995 USENIX Security Symposium [1].

The requirements for a system security assessment and compliance tool are as follows:

- Minimally, the tool must check for the conditions which allow exploitation by *AutoHack*. Additional security configuration checks are desirable.
- The solution must be easy to use and support both interactive and non-interactive use. In interactive, verbose mode, the security assessment and compliance tool can be used for training systems administrators in network and system security. The non-interactive mode is required for automating security administration.
- The tool must function on every version of the Sun Operating System from SunOS 4.0

through Solaris 2.4. There must be no dependencies on languages, tools, or utilities that are not included in the core distribution of the operating system.
- The security assessment and compliance system should ideally be modular, simplifying the modification of existing modules or the addition of new modules.
- Policy based, hands off operation is desirable. The tool must be able to facilitate compliance with enterprise computing security standards.

System Overview

Take One – *AutoHack*

The first release of our system security assessment and compliance solution was a shell script that I wrote one weekend. This quick and dirty prototype, *autohackfix*, was a single interactive script that addressed only those security problems that were identified by *AutoHack*. It was designed to be run interactively by a systems administrator on individual systems and checked for the conditions on a system that would allow *AutoHack* to exploit it. If a condition existed, the systems administrator was given the option to fix the problem automatically or not.

The *autohackfix* script was obviously limited in scope and was replaced within a couple of weeks by *SunSWAT* – the Sun Security Weakness Attack Tool.

Take Two – *SunSWAT*

SunSWAT, in its current release, is a series of scripts and programs designed to fix various security holes in Sun systems running the Solaris operating system (either Solaris 1.x or Solaris 2.x). *SunSWAT* consists of a single driver script and several modules.

Driver Script

The driver script, *sunswat*, is used to determine system configuration, check for prerequisites and call the individual modules. Usage for *SunSWAT* is:

```
# sunswat [-a] [-v] [-h] [-c file]
```

where:

-a is an optional flag that forces the default response for all questions. The default responses may be over-ridden by modifying the *$HOME/.sunswatrc* file or by specifying an

```
echo
echo "Checking rexd ... "
CheckRootOwn $INETD
CheckWorldWritable $INETD
CheckGroupWritable $INETD
if $GREP "^rexd" $INETD > /dev/null; then
        if [ "$VERBOSE" ]; then
                echo
                echo "The rexd daemon is a notoriously insecure service"
                echo "which would allow people worldwide to steal any"
                echo "file on your system.  It should be removed or disabled."
                echo
        fi
        AskYN " Disable rexd?" ${REXD_DISABLE:-"Yes"}
        if [ $? -eq 1 ]; then
                $SED 's/^rexd/#rexd/g' $INETD > $INETD.$$
                $MV $INETD.$$ $INETD
                $CHMOD 644 $INETD
                KillPID inetd
                (echo "$SED 's/^rexd/#rexd/g' $INETD > $INETD.$$") >> $TF
                (echo $MV $INETD.$$ $INETD) >> $TF
                (echo $CHMOD 644 $INETD) >> $TF
                echo "AH0:$INETD:rexd:disabled" >> $AF
        else
                echo "AH4:$INETD:rexd:enabled" >> $AF
        fi
else
        echo "AH0:$INETD:rexd:disabled" >> $AF
fi
```

Figure 1: Sample *autohackfix* Code

alternate configuration file using the **-c** option.

-v is an optional flag that causes *SunSWAT* to run in verbose mode, explaining why conditions should be changed to enhance security.

-h is an optional flag that displays a help message.

-c file can be used to specify an alternate configuration file.

If no options are given, *SunSWAT* is run interactively. The configuration file, *sunswatrc*, can be used to enable or disable the different modules and to change the default response for any or all "fix this?" questions.

After executing each of the individual modules, *SunSWAT* sends a report containing information on what it found, and what it did to correct security problems, back to a central tracking server. The information is processed automatically and a tracking database is updated. This database is used to generate daily progress and escalation reports.

Modules

The *SunSWAT* driver script calls the following modules:

- **autohackfix.sh** repairs the problems identified by *AutoHack*. "Web of trust" checking is done, by investigating the contents of the */.rhosts* and */etc/hosts.equiv* files, optionally removing insecure entries such as "+" or comments. Entries for insecure network services can be commented out of the *inetd* configuration file. Piped mail aliases are verified and optionally disabled. Basic password file checking is done, with comments being removed and accounts with no passwords disabled.

  Figure 1 shows an excerpt from the *autohackfix* module that is used to check and fix the configuration of *rexd* in the */etc/inetd.conf* file. The variable *$TF* represents the name of a trace file that is kept for each *SunSWAT* run. The *$AF* variable is used to represent the name of an audit file which is sent to a central server which is used to track *SunSWAT* usage and update a system status database.
- **fileperms.sh** is used to change the file permissions of system files to a more sane mode (from a security standpoint).
- **ftp.sh** checks for, and optionally disables FTP for root and other non-human users and then proceeds to verify the configuration of anonymous FTP on the system if it is enabled. Ownership and permissions of all anonymous FTP directories and files are verified and optionally corrected.
- **misc.sh** is a script to check and fix several miscellaneous security problems, for example, checking and removing writability of important system directories, checking and modifying the permissions of */etc/utmp*, and searching and modifying permissions of setuid/setgid scripts.

```
#################################################################
# /etc/inetd.conf (or /etc/inet/inetd.conf):
#
# TFTP_DISABLE  Disable tftp completely?
# TFTP_SECURE   Configure tftp in secure mode?
# UUCPD_DISABLE Disable uucpd?
# REXD_DISABLE  Disable rexd?
# SYSTAT_DISABLE        Disable systat?
# NETSTAT_DISABLE       Disable netstat?
# FINGERD_DISABLE=      Disable fingerd?
# RUSERSD_DISABLE=      Disable rusersd?
# SPRAYD_DISABLE=       Disable sprayd?
#
#################################################################
TFTP_DISABLE="No"
TFTP_SECURE="Yes"
UUCPD_DISABLE="Yes"
REXD_DISABLE="Yes"
SYSTAT_DISABLE="No"
NETSTAT_DISABLE="No"
FINGERD_DISABLE="No"
RUSERSD_DISABLE="No"
SPRAYD_DISABLE="No"
```

Figure 2: Excerpt from *SunSWAT* Configuration File

- **nfs.sh** checks the configuration of NFS and modifies the */etc/exports* or */etc/dfs/dfstab* file as needed.
- **patch5x.sh** is a script to apply security patches to Solaris 2.x systems. A list of mandatory security patches is maintained by Sun's Network Security Group and distributed with the *SunSWAT* program.
- **root.sh** checks and fixes root's environment, ensuring that there is no "." in root's path, verifying a sane value for root's umask and checking and optionally modifying the permissions of directories in root's path.

Configuration File

The *SunSWAT* configuration file can be used to predefine directory locations, select which modules to run and how to respond to all questions. The configuration file can always be overridden at run time if *SunSWAT* is invoked in interactive mode.

Figure 2 shows a sample set of rules for *inetd* configuration. The systems administrator can modify the default behavior of *SunSWAT* by following the directions in the configuration file and basically answering a set of "yes" and "no" questions. Several sample configuration files are provided for the systems administration staff representing different security models. For example, there is a configuration file that provides for adherence to the enterprise desktop system security standard. Others are provided for different production servers and other types of hosts on the network.

Initial Implementation

SunSWAT was designed to help solve the problem of securing the sheer volume of systems on Sun's internal network. It is distributed via NFS mounts using Sun's internal software distribution mechanism and is available on every system on Sun's network. Superuser privilege is required to execute *SunSWAT*, since it modifies system files owned by root.

SunSWAT was used in the initial effort to secure all systems on Sun's networks, most frequently executed from the command line by qualified systems administrators.

Systems administrators further automated the use of *SunSWAT* by centrally controlling its execution from a single trusted host within an administrative domain. Configuration files specific to a domain's user requirements are developed and used

```
#!/sbin/sh
#
if [ -f /usr/dist/local/config/pkgs/sunswat/samples/desktop_ens_sunswat_rc ]
then
        SWAT_RC="/usr/dist/local/config/pkgs/sunswat/desktop_ens_sunswat_rc"
elif [ -f /usr/dist/config/pkgs/sunswat/samples/desktop_ens_sunswat_rc ]
then
        SWAT_RC="/usr/dist/config/pkgs/sunswat/samples/desktop_ens_sunswat_rc"
elif [ -f /usr/dist/pkgs/sunswat/samples/sunswatrc.autohackfix ]
then
        SWAT_RC="/usr/dist/pkgs/sunswat/samples/sunswatrc.autohackfix"
else
        exit
fi

if [ -x /usr/dist/pkgs/sunswat/bin/sunswat ]
then
        run_level='/usr/bin/who -r | /usr/bin/awk '{ print $3 }''
        if [ "${run_level}" -gt 2 ]
        then
        echo "Executing sunswat"
                ( /usr/dist/pkgs/sunswat/bin/sunswat -a -c ${SWAT_RC} ) &
        else
                echo "Queuing sunswat job to be executed at midnight"
                echo "/usr/dist/pkgs/sunswat/bin/sunswat -a -c ${SWAT_RC}" | \
                        /usr/bin/at -s midnight
        fi
fi
```

Figure 3: *SunSWAT* Startup Script

to control execution of *SunSWAT* from a central location.

Further Implementation

The *SunSWAT* tool and how it is used continues to evolve.

System Security Standardization

SunSWAT has been selected by the Sun computing standards development group as the method to be used to comply with existing and evolving security standards and policies. In the method prescribed by the standard organization, the *sunswat_startup* script, located in */etc/init.d*, is executed during each system reboot. The script, shown in Figure 3, is used to schedule *SunSWAT* to run at midnight following the system boot. If the startup script is started from the command line, at any time other than a system reboot, *SunSWAT* is run immediately.

On existing systems, this *sunswat_startup* script is put into place using either *rcp* or *rdist* from a central administrative trusted host. On new installations and system upgrades, this *sunswat_startup* script is put into place by the Solaris installation process. Most system administration groups at Sun are now using the AutoInstall process for system installations and upgrades and the *sunswat_startup* script installation has been included in the finish scripts of this automated process. For those groups not using AutoInstall, the installation of the *sunswat_startup* script has been included in the system installation procedural documents produced by the computing standards development group.

This model works well for complying with changing security policies. Global security policy changes are made on a centrally located configuration file which all clients reference. Each system is brought into compliance upon its next reboot. If there is some urgency in applying a new security patch, for example, the systems administrators can use the remote execution of *SunSWAT* from a central administrative host method.

System Auditing

Elementary auditing is available with the current version of *SunSWAT*. A standalone script, *swataudit*, can be used to simulate an *AutoHack* attack attempt on the system. This is used by systems administrators to verify the security fixes that they have made at any time without having to wait for the next *AutoHack* sweep. On critical servers and on some desktop systems, *swataudit* is run daily from *cron* which provides for simple auditing and intrusion detection.

More extensive auditing and escalation features are being designed into a later version of *SunSWAT*.

Evolution Of *SunSWAT*

SunSWAT has been used to make modifications on more than 4100 systems on Sun's internal networks and has become widely accepted as the primary method for security policy enforcement. Due to this fact and based on the feedback provided by the systems administrators responsible for applying security measures to systems, *SunSWAT* Version 2 has been designed and is currently being developed and tested. Several new features have been added to *SunSWAT* Version 2.

SunSWAT Version 2 Features

Modularity has been extended. Security checks are grouped logically and many new checks have been added. The functions formerly combined into *autohackfix* have been moved into the other modules and grouped logically by function. The *autohackfix* module still exists (as it is used quite extensively), however, it simply calls the appropriate functions from the other modules.

New modules and modules which have changed significantly are as follows:

- **trust.sh** handles the "Web of Trust" checking formerly done by the *autohackfix* module and extends it to include checking users' *.rhosts* files.
- **inetd.sh** is used to check for insecure services or configurations in */etc/inetd.conf* or in the */etc/inetd/inetd.conf* file.
- **mail.sh** performs all sendmail patching and all checking and repairing in the */etc/mail/aliases* file.
- **nis.sh** checks NIS maps for insecure configurations and notifies the appropriate support group for resolution. No automation of the fixes is done at this time.
- **patch4.x** has been added to install mandatory security patches on SunOS 4.x systems.
- **passwd.sh** has been extended significantly to perform several new checks and fixes within each of the files: */etc/passwd*, */etc/shadow*, and */etc/group*.

The **driver script** is being replaced by a C program that controls all of the modules. Checksums for each of the modules are to be coded into the driver program so that the user can be assured that modules have not been tampered with.

The **user interface** has been modified so that the modules to be run can be specified on the command line. For example, in the new scheme, the following command:

```
# sunswat -a autohackfix passwd
```

would cause *SunSWAT* to execute in automatic mode, calling only the *autohackfix* and *passwd* modules. If *SunSWAT* is called without the **-a** switch, a menu is displayed which allows the user to select which modules to execute.

The scheme used for patching systems has been extended and redesigned. Dependence on a single patch distribution server has been removed and the entire process is now much more distributed. A new module for applying patches to SunOS 4.x systems has also been added.

An *undo* feature is being developed which allows the user to restore the system to the state that it was just prior to running *SunSWAT*.

Future Plans

As new exploits become known, *SunSWAT* will be modified to correct for them. Similarly, as new security features get added to the operating system, *SunSWAT* will change to keep pace.

SunSWAT will be extended to be used as a general auditing tool, with the ability to generate either machine readable or human readable audit reports. Additional functionality in the areas of intrusion detection and integrity checking is currently being researched. We are considering developing a graphical user interface, possibly using a WWW browser, to facilitate configuration and centralized operation.

We are also designing a new tracking and reporting scheme, using a commercial relational database, which will allow us to implement more sophisticated escalation procedures. This database will be tied into the system registration and human resources databases to provide information on system ownership and management chain. The escalation process is simple – the first time a system appears on the list, the system owner will receive notification that the system is out of compliance with corporate security policies. If the system is identified by *AutoHack* as having the same types of security weaknesses the following month, notification is sent to the system owner's manager. On the third consecutive violation, notification is sent to an operating company information resources executive. After this final notice is issued and the system is still in violation of the security policies, it is to be removed from the network.

Conclusions

Security administration in large open networks is a very challenging task and it is imperative to automate as much as possible. Most of the tools available today focus on the identification of security vulnerabilities, not the correction of conditions and configurations that cause a system to be vulnerable. A tool that we designed to address security weaknesses identified by network security audits was easily extended to facilitate standardization and adherence to Sun internal security and system configuration policies.

Availability

SunSWAT is only available for use within Sun Microsystems Incorporated.

Acknowledgments

Thanks to all members of the Network Security Group at Sun for providing ideas for modules and code review. Special thanks to Casper Dik for providing much of the code for the file permissions module and to both Casper and Alec Muffett for developing system exploit code which has been adapted for *SunSWAT*. Many thanks to Tom Jollands for developing the AutoInstallation and *sunswat_startup* scripts and for spear-heading the system standardization efforts. Thanks also go to Karen Doby for encouraging me to write this paper and for her review of it.

Author Information

Karen Casella is a Network Security Engineer for the Network Security Group at Sun Microsystems Incorporated in Mountain View, CA. She has taken the roundabout path to security engineering, beginning her professional career as a mechanical engineer, migrating into systems administration and finally making the move into the exciting field of network security. Reach her via U.S. Mail at Sun Microsystems, Inc.; 2550 Garcia Avenue, Mountain View, CA. Reach her via electronic mail at karen.casella@eng.sun.com.

References

[1] Muffett, Alec, "WAN Hacking with *AutoHack*: Auditing Security Behind the Firewall", Proceedings of the 5th USENIX Security Symposium, June, 1995.

[2] Farmer, Daniel and Venema, Wietse, SATAN (Security Administrator Tool for Analyzing Networks) documentation, April, 1995.

[3] Farmer, Daniel and Spafford, Eugene H., "The COPS Security Checker System", Proceedings of the Summer 1990 USENIX Conference, pp. 165-170, June 1990.

[4] Safford, David R., Schales, Douglas Lee and Hess, David K., "The TAMU Security Package: An Ongoing Response to Internet Intruders in an Academic Environment", Proceedings of the Fourth USENIX Security Symposium, 1993.

[5] Garfinkel, Simson and Spafford, Gene, Practical UNIX Security, O'Reilly & Associates, Inc., 1991.

Multi-platform Interrogation and Reporting with Rscan

Nathaniel Sammons – Colorado State University

ABSTRACT

This paper describes Rscan – a tool that allows a System Administrator to easily write and execute scripts on a single machine or an entire network of machines. Rscan can be used to automate security checks, configuration checks and many other tasks. If a module (a collection of scripts) is written with different sections for different operating systems and different versions of each operating system (as they are intended to be written), Rscan will automatically select the correct set of scripts to execute on any particular operating system.

As Rscan runs it generates reports which can be written as either plain ASCII or HTML files. Rscan is specifically designed to be run in a heterogeneous environment and is easily adapted to new and different operating systems. Two running modes are available: text mode and GUI mode. When running in text mode, all output that is generated is sent to either standard out or standard error. When running in GUI mode, Rscan starts a WWW browser such as Netscape Navigator or NCSA Mosaic and the user can configure the system, run scans and read reports using the browser as a GUI.

Motivation

There have long existed programs like COPS [1] and TAMU [3] that are capable of scanning an individual computer for a set of security holes, but they have traditionally been limited to running generic scans that look for problems that occur in the "general case." These utilities have historically not included code that examines a system for security holes that are unique to a particular operating system, nor have they included a way to easily integrate custom scans, or to automatically select specific scans based on the operating system of the machine being scanned. COPS alluded to the fact that a system able to automatically select which scans to run based on the operating system of the machine being scanned would be a step in the right direction, but did not offer any solutions.

There have also been applications such as `rdist` that are designed to quickly and easily distribute files to many hosts on a network. There has not, however, been a tool that is specifically designed to run in a heterogeneous environment and execute complicated scripts on many machines and to execute different scripts under different operating conditions.

Rscan is intended to fill this void. It offers a uniform way to run any number of independent scans on any number of computers and organize the results of all the scans in a clean, formatted report. Rscan will also automatically select the appropriate scan for any operating system provided that a module has been written with specific code for that operating system. It also provides a straightforward, uniform method of selecting which modules to execute on a particular system.

Designing Rscan

Rscan started out as an IRIX-only security utility, but has outgrown those bounds and become a modular, extensible interrogation tool. In rewriting the original code, the following design goals were considered:

1. The solution must not be overly cumbersome or its obfuscation will overshadow its usefulness.
2. Setup and configuration must be simple and logical.
3. The system must be as robust and fault-tolerant as possible.
4. Writing modules must be a straightforward process and there should exist an API to help facilitate this.
5. Reports must be nicely formatted for ease of reading, and there should be a uniform method to write reports to help keep them organized.
6. The interface must be simple and functional and not get in the way of actually getting work done.

Software should bend over backwards to meet the user's needs, not visa versa, so a lot of effort was put into making Rscan robust enough to keep users from needing to modify the code to meet their needs. Almost every aspect of Rscan can be customized through the use of command line options and/or configuration file options.

Rscan was originally intended to be used only for security checking, and currently all of the publicly available modules do just that, but Rscan is robust enough to provide functionality far beyond security checks. Rscan can be used to implement

custom scripts for a site that check for proper configurations and perform other tasks that have primarily been done manually in the past. For instance, a module can easily be written to check that a set of utilities has been installed on a new machine, or that the utilities that are installed have not been tampered with, etc.

An Rscan module can have both operating system independent and operating system specific parts to it. If the module is selected for execution on a particular machine, then all the OS-independent scans will be run, as will any OS-specific scans. For every module, there is an OS-independent initialization script that sets up any variables, functions, and other data that the module will need to run under any operating system. For each OS-specific set of scans in a module, there is a corresponding OS-specific initialization script that can set up other, more specialized information that the module will need to run (such as the path to a precompiled MD5 checksum utility, etc). Within an OS-specific section of a module, there is a set of scans that run on all versions of that operating system and a set that run only on a specific version of that operating system (e.g. 4.0.5 or 5.3 under IRIX or 4.1.1 or 4.1.3 under SunOS). All, some or none of these sections need to be present for the module to function, as Rscan will automatically decide what to run. A module could easily be written that runs the proper copy of Crack or Tripwire for a particular operating system on a network of hundreds or thousands of hosts, summarizing its results in a central report as it runs.

Rscan can be used to encapsulate the functionality of any tool that only runs on one machine at a time by writing a simple "wrapper" script (which could be as simple as starting a subprocess and relaying everything it says using the `&report()` API function) that runs the tool on the remote machine. Trivial modifications can be made to a module so that the right, often operating system specific, command line arguments are passed to the tool when it is run on the remote machine. This is Rscan's most attractive feature: making it easy to write a tool that runs on all the operating systems in an organization, and can be run from a central location with the touch of a button.

The design incorporates a flexible reporting mechanism that can generate a series of reports (one for each host), or a single report for the entire network. A set of functions to be used by all modules to report progress and test outcomes to the screen and to write data to the report file is provided. Reports can be written in either plain ASCII or in HTML. Rscan will automatically properly reformat data based on whether it's writing an ASCII or HTML report.

General Implementation

Rscan is implemented in perl [2] for many reasons. Perl is widely regarded as the de-facto system administration language, and it provides many attractive features, especially when designing for multi-platform use. Since the code for Rscan does not use any of the new functionality provided in perl 5, it will run under either version 4 or 5 of perl.

When scanning, each module is run in its own isolated process. Inside each module, every individual scan is also run in its own process. This is done to ensure that every module can do anything it wants to (including modification of system-defined variables, etc) and not have to worry about how its actions affect the other modules. Variables can be changed and anything can be done inside a scan without worrying about how it will affect other scans in that module or other modules that will be run after its run has been completed. In addition to these benefits, if a scan or an entire module crashes, it will not abort the entire scanning run. In the case that a scan in a module or an entire module exits abnormally (with an exit value other than 0, usually indicating a crash), Rscan will log to the report that the process exited abnormally, giving the scan or module that was running, its process ID number, and its exit value.

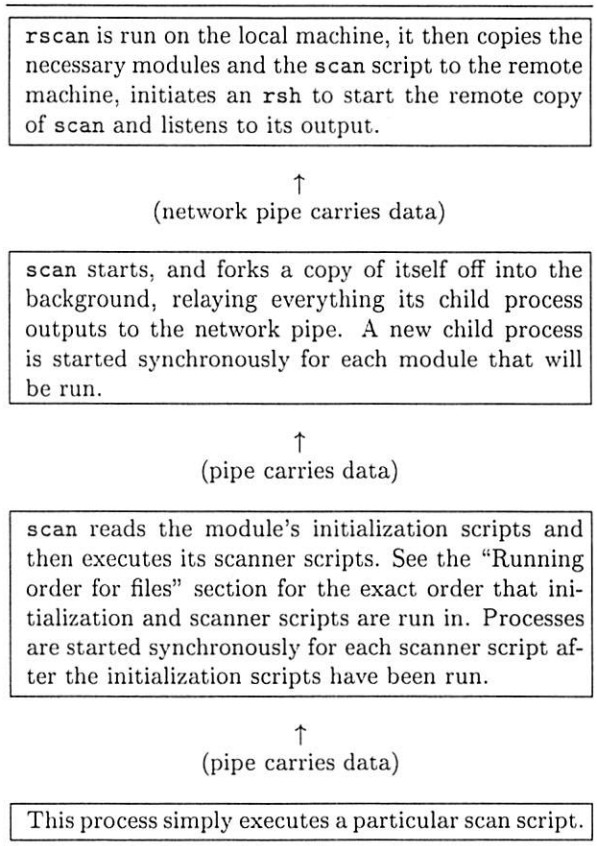


Figure 1: Typical Rscan process structure

Rscan is composed of three scripts, named "rscan," "scan" and "modman." rscan is run on the machine doing the scanning of the other hosts, and acts as a supervisor for the actual scanning script, scan. rscan copies scan and the necessary modules to the host that will be scanned. scan is then executed by rscan, and it sends data back across a UNIX pipe. Each module is executed in much the same way on the remote machine. Scan forks a copy of itself off and that second copy then runs any initialization scripts for the module it will be running (both OS-independent and OS-specific ones), then forks a copy of itself off to run each scan in its module. Each process is connected to its parent by a UNIX pipe. Figure 1 shows the typical process structure of an Rscan run.

The third script, modman, is a *MOD*ule *MAN*agement utility that simplifies installing new modules and other module administration tasks. Modman installs, removes, and backs up modules and provides an easy way of tracking installed modules.

GUI Implementation

When running in GUI mode, Rscan starts two very simple HTTP [4] servers that each listen to a unique port on the machine acting as the server. Rscan generates a 30 character session key by viewing the current network statistics (obtained from the output of netstat) and by examining the current process table. This session key makes it much harder (though not impossible, as with any key) for another host to interact with Rscan remotely. Rscan then starts up a WWW browser (Netscape Navigator is recommended, but others will work) with the URL that points to the first HTTP server.

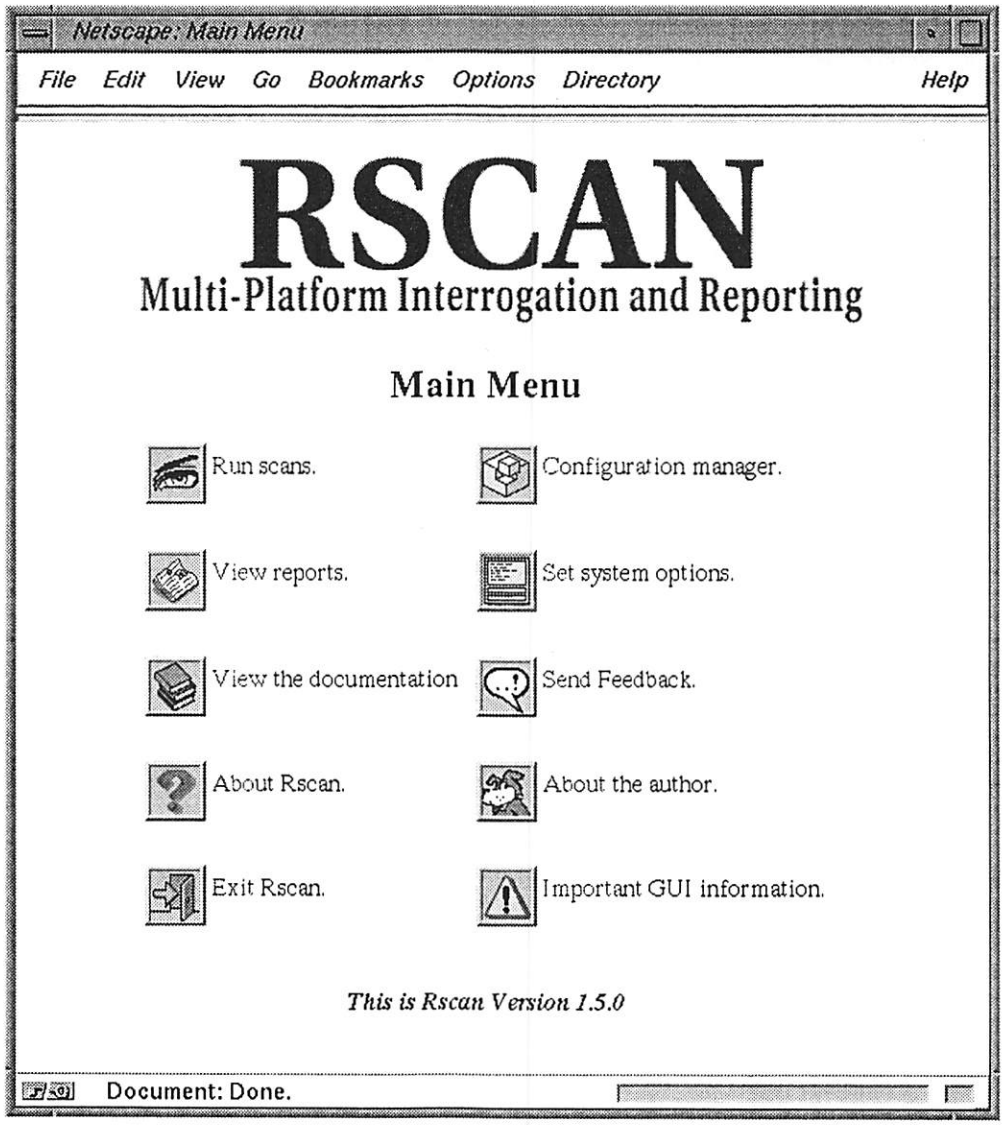


Figure 2: Main Menu in GUI Mode

One of the HTTP servers only serves images and the other only serves documents and controls scans. This is done so that while a scan is running, and thus tying up the document server, inlined images can still be inserted into the output of the document server.

When Rscan starts up in GUI mode, it displays the main menu, which is shown in Figure 2. From this menu, the user can choose to run scans, edit configuration files, view reports, set system options, view documentation, send feedback to the author, view the Rscan home page, read about the author, exit Rscan or view special documentation about running in GUI mode.

Using a WWW browser as a GUI presents one glaring problem. Most WWW browsers now implement the `Referrer:` field in the HTTP request header, which will disclose the session key to the server which holds the next URL being accessed. If a user chooses to jump directly to a new URL that is outside of the servers that Rscan is running by selecting the "Open URL" menu or its equivalent, there is no way to keep the session key from being disclosed, and there is no way for the servers run by Rscan to know that the user chose to open a URL that's outside of the Rscan servers – this is simply a fact of how HTTP and WWW browsers function internally. Because of this, users are *strongly* advised not to jump directly to a URL while running Rscan. Occasionally, it is necessary to present the user with a URL in a hypertext reference that points outside of the Rscan servers (this happens frequently when viewing reports, and even in some of the main menu options). When this is done, Rscan translates that URL into a Referrer screen page. The Referrer screen page notifies the user that they are about to jump outside the confines of the Rscan servers. The user is presented with a hypertext reference that will cause Rscan to shut itself off while leaving the WWW browser running. If the user chooses to follow the reference, then Rscan shuts down the document and image servers, and prints a final HTML page that tells the user that the servers have been shut down and they will not be able to do more scanning until they restart Rscan. At the bottom of the page is a hypertext reference that points to the original URL that is outside of the Rscan servers. This may seem like a lot of work to go through, but it ensures that any URL that is presented to the user will not compromise the security of the Rscan servers.

Once the `Referrer:` problem has been dealt with, WWW browsers provide one great advantage over traditional GUI methods: They are very portable. All that needs to be available on a prospective platform for Rscan to run is a version of perl that supports the use of UNIX sockets and a WWW browser. Although this does not let Rscan take advantage of some of the features of, for instance, the TkPerl implementation, it is much easier to port to a new operating system. Because of the simplicity of the HTTP specification, writing an HTTP server to act as a core for a GUI is not particularly hard, and since most browsers support the FORMS standard, input and interaction with the user is clean and the interface is aesthetically pleasing.

Using Rscan

Rscan was designed to be easily used by System Administrators of almost all skill levels. It can be run in two basic modes: regular (text) mode, and GUI mode. Quiet mode is just like text mode, but standard out is closed so that no progress information is written while scans are being run. In text mode, progress information is written to standard out while Rscan is running. In GUI mode, as discussed previously, a WWW browser is used as an interface and the user can configure Rscan, read reports and run scans.

Configuration

All command line options (except for `-gui` and `-guidebug`) are only available when not running in GUI mode. Below is a list of command line options and the function of each.

-ascii Forces the writing of ASCII reports (which is the default). This is used to override the `html reports` configuration file option in the configuration file being used.

-cf filename Uses a different configuration file than the default, which is "`config/default.cf`." If a relative pathname is supplied, then it is taken to be relative to inside the `config` directory, if it's an absolute path, then its relative to the root directory.

-config This enters the text-based interactive configuration file editor, from which users can move, rename, delete, edit, and (most importantly) test configuration files. The path to the WWW browser to be used with the `-gui` and `-guidebug` options can also be set.

-debug Turns on (copious) debugging output while in text mode. This is handy if users are having a problem or a scan is not working properly.

-gui Causes Rscan to run in GUI mode. This option is special in that if it is used, it must be the *only* option used.

-guidebug Like the `-gui` option, this causes Rscan to run in GUI mode, but it also turns on debugging output like the `-debug` option. This option is special in that if it is used, it must be the *only* option used.

-h This prints a list of command line options, short descriptions of each, and information on their usage.

-html Forces the reports to be written in HTML. If reports are written in HTML, report filenames have a `.html` added to them (although if a user specifies a reportname that ends in `.html` then

another `.html` will not be added to the end).

-list Lists all installed modules by short and long name, and also prints the module's version number. Every scan in each module is also listed with a brief synopsis of what its function is.

-local Runs the scans on the local machine only. If a user wants to specify a modlist for the local machine and there is not a default one set in the configuration file, or the user not using a configuration file, they should specify the modlist using the `-modlist` option.

-machine Specifies a set of machines to scan and associates a location for perl and a modlist with each. Users can have as many `-machine` options as they like, and conflicting machine definitions will be resolved in the order they appear: the last definition for a machine sticks. The format for this option is as follows:
`machine` *host1,...,hostN /path/to/perl modlist*
where *host1,...,hostN* is the set of hosts to scan, */path/to/perl* is the path to the perl executable on the machines or the word "`same`" if perl is located in the same place as perl that is running the Rscan process, and *modlist* is the modlist to associate with each host. Users can give a modlist of "`any`", in which case all applicable modules and scans in those modules will be run.

-modlist modlist Sets a global modlist for all scans. This will override options set in the configuration file, and all modlists given on the command line (in `-machine` options).

-nocf Don't even look at any of the configuration files. If this is not given, then the default configuration file is used (which can be overridden by the `-cf` option).

-quiet Turns off all printing to standard out. This is good if a user is running Rscan from a crontab file and don't want to get mail all the time about it running. Reports are still written.

-reportdir directory Specifies an alternate report directory. If it is a relative path (does not start with the "/" character), then it is taken to be relative to the directory that Rscan is in, otherwise it is taken to be an absolute path.

-reportname format Specifies the report naming format to be used. Report name formats are a string of text (whitespace and the % character are not allowed) that can use the substitutions shown in Table 1. The default reportname format is:
`rscan.%H-%M.%D.%Y-%h:%m:%s`
Because some of these characters are considered meta-characters by some shells, it is recommended that a report name format be enclosed in "double quotes" when issued on the command line.

-separate Writes a separate report for each host that is scanned. The %H in the report name format is replaced with the name of the host (which is the word "`network`" if separate reports are not being written).

-single Forces the writing of a single report. This is used to override the `separate reports` option in a configuration file.

-tmp directory Uses the given directory as the temporary directory when running scans on remote machines. The default is `/tmp`.

%I	Integer time (seconds since Jan. 1, 1970)
%H	hostname or "network"
%M	month number
%D	day of the month number
%Y	two-digit year number (ex "95")
%h	hour when Rscan was started
%m	minute when Rscan was started
%s	second when Rscan was started

Table 1: Substitutions for -reportname

Configuration files are basically a way of collecting command line options together for reuse. The default configuration file is `default.cf` located in the `config` directory. Other configuration files can be selected by using the `-cf` *filename* command line option, as described above. In a configuration file, options are given one per line. Blank lines and lines beginning with the "#" character are ignored. Options can be continued over multiple lines by using the "\" character at the end of a line. If a line is continued, then all whitespace after the \ and all whitespace at the beginning of the next line is deleted. Configuration file options are listed below.

html reports Writes reports in HTML. If reports are written in HTML, report filenames have a `.html` added to them (although if the user specifies a reportname that ends in `.html` then another `.html` will not be added to the end).

machine Specifies a set of machines and associates a location for perl and a modlist with each. Users can have as many `machine` options as they like, and conflicting machine definitions will be resolved in the order they appear: the last definition for a machine sticks. The format is as follows:
`machine` *host1,...,hostN /path/to/perl modlist*
where *host1,...,hostN* is the set of hosts, */path/to/perl* is the path to the perl executable on the machines or the word "`same`" if perl is located in the same place as perl that is running the Rscan process, and *modlist* is the modlist to associate with the hosts. If the modlist is left out, a modlist of "`any`" is assumed.

modlist modlist Sets a global modlist for all scans. This will override options set in the modlist field for each machine definition in the configuration file.

reportdir directory Specifies an alternate report

directory. If it is a relative path (does not start with the "/" character), then it is taken to be relative to the directory that Rscan is in, otherwise it is taken to be relative to the root directory.

reportname format Specifies the report naming format to be used. See the description of the `-reportname` command line option for how to write a reportname format.

separate reports Writes a separate report for each host that is scanned. The %H in the report name format is replaced with the name of the host (which is the word "`network`" if separate reports are not being written).

tempdir directory Uses the given directory as the temporary directory when running scans on remote machines. The default is `/tmp`.

Editing Configuration Files

Configuration files can be edited in several ways. They can be edited by hand using a text editor such as `vi` or `emacs`. They can be edited by invoking Rscan with the `-config` option, which invokes a rudimentary text-based interactive configuration file editor, or they can be edited from within the GUI.

When edited from within the GUI, the user can edit the file by hand in the WWW browser's window or use the point-and-click editing facilities the GUI provides to edit the file. The user is presented with text input boxes, menus and checkboxes to select options as shown in Figures 3, 4, and 5.

From the main editor panel shown in Figure 3, the user can go onto the Machine Definition Lister, as shown in Figure 4. From there, the user can delete, create new, and edit existing machine definitions for the selected configuration file.

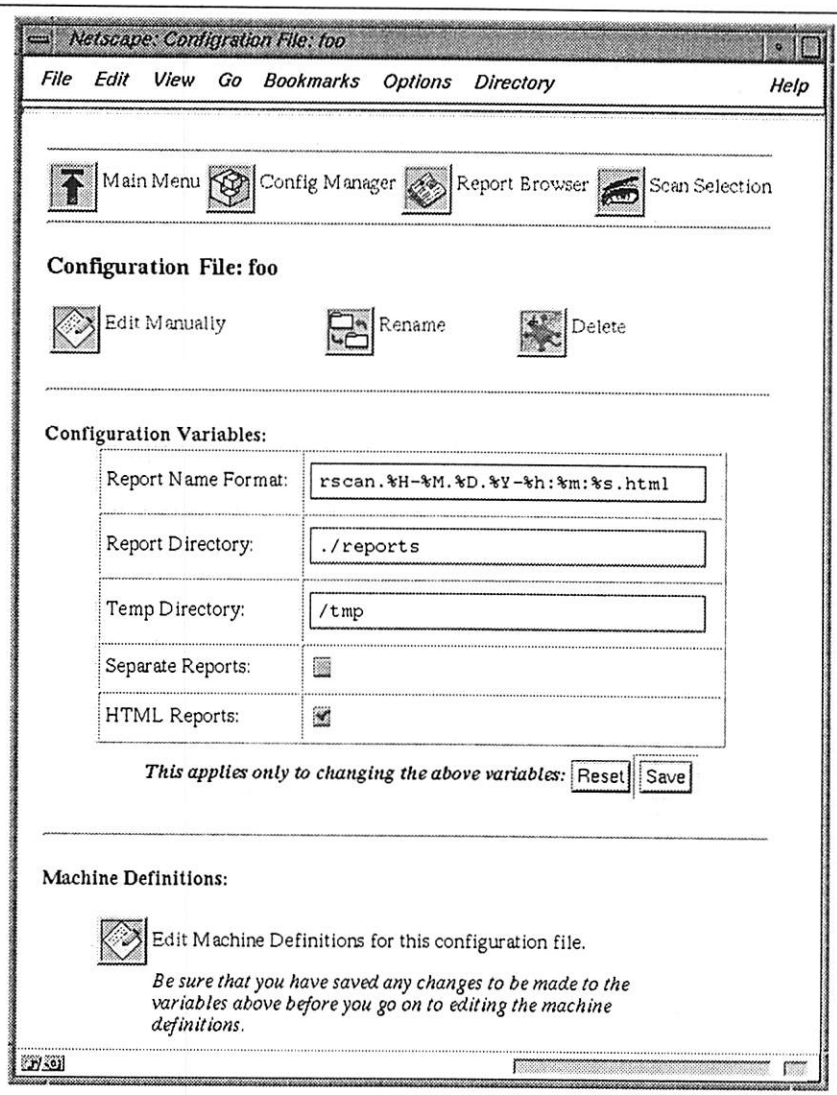


Figure 3: Initial GUI Configuration Editor

When the user chooses to edit a machine definition, they will see a panel like that in Figure 5. On this panel, the user can add hosts to the list of machines that the machine definition affects, edit the location of perl for the set of machines, and edit the modlist for the set of machines. When editing the modlist, the user is presented with a list of installed modules, and all they need to do is click on toggle buttons to select and de-select modules to run. The user can also choose to run all the scans in the module, only selected scans that are chosen from a scrolling list, or all scans in the module *except* the ones that they have chosen. When finished selecting scans to run and other options, the user simply needs to click on the "Save" button to have the machine definition saved to the configuration file.

Modlists

One of the most powerful features of Rscan is the concept of the modlist. A modlist (*MOD*ule *LIST*) is a way of specifying which modules will be run on specific hosts and which scans in a particular module will be run. It allows the user to be very picky about selecting what to run on any machine.

A modlist may be as simple as the word "`any`," which instructs Rscan to run all applicable modules and scans in those modules when scanning the remote machine. It can also specify that out of module `foo` we want to run only scans `x`, `y`, and `z` and that in module `bar` we want to run all scans *except* the `i`, `j` and `k` scans.

The mechanism for defining modlists is simple and straightforward, and a modlist can be applied to any number of hosts without the need of writing it many times (since they can become long and relatively complicated).

Module names are separated by commas, with no space in between anything. For instance, a modlist of "`ThisModule,ThatModule,TheOtherModule`" tells Rscan to run the `ThisModule` module, the `ThatModule` module and the `TheOtherModule` module.

Modlists can be more complicated and more powerful than this. Using the `-list` option to Rscan, users can get a list of modules that are installed and a list of the individual scans that compose each module. Using these lists, usrs can construct a custom subset of scans to run from each module.

Let's say that the `ThisModule` module had only the scans named `sendmail`, `rdist`, `rhosts`, and `xsession`. Then if a user only wanted to run the `sendmail` and `rhosts` scans out of that module, they could replace the `ThisModule` part of the previous modlist with the following:

```
ThisModule[sendmail:rhosts]
```

and only those scans would be run. The scanner names are the names of the `.pl` files without the `.pl` extension. Users can also get the same result if they use the negation operator ("`-`") in the modlists.

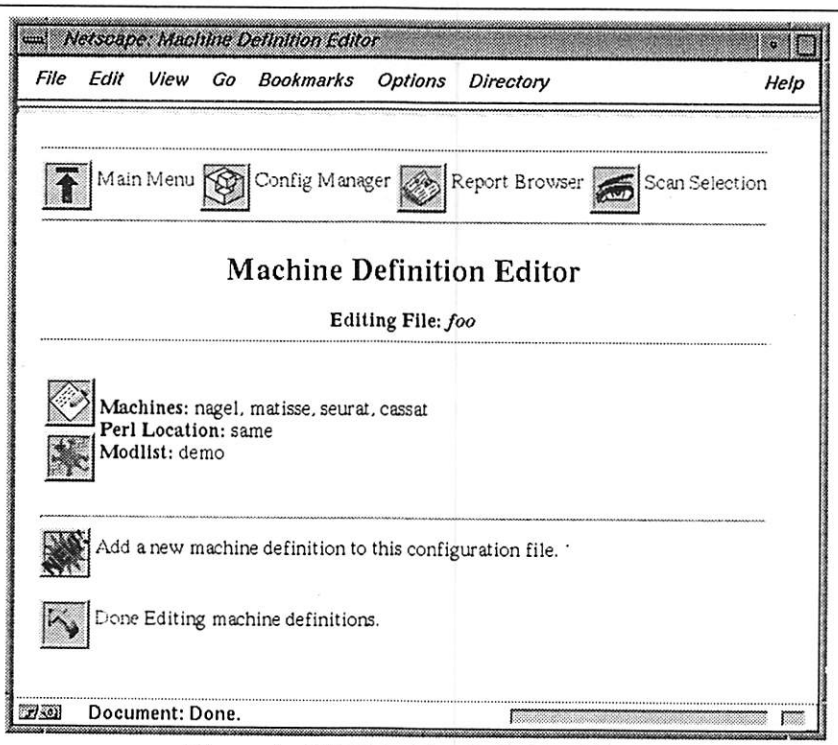


Figure 4: GUI Machine Definition Lister

Because in this example we assume there are only those four scans in the module, the above is equivalent to:

```
ThisModule[-rdist:-xsession]
```

This form of a modlists tells Rscan to run everything in the `ThisModule` module *except* the `rdist` and `xsession` scans.

If specifying scans in a modlist, be sure not to mix the two kinds when specifying what scans to run in a particular module. A modlist cannot contain both regular and negated scan selections. For instance, the modlist

```
ThisModule[-rdist:xsession]
```

is invalid, and would be caught by Rscan's modlist parser, but

```
ThisModule[-rdist:-xsession],
    TheOtherModule[thisscan:thatscan]
```

is valid because it does not mix the two kinds in a single module definition.

A modlist has no limit on length, though the user's shell may impose one when they are used on the command line. Users can specify as many modules and as many scans as they want, provided that the space character doesn't show up, and module names are separated with the ',' character and scan names with the ':' character. A modlist may need to be protected with "double quotes"

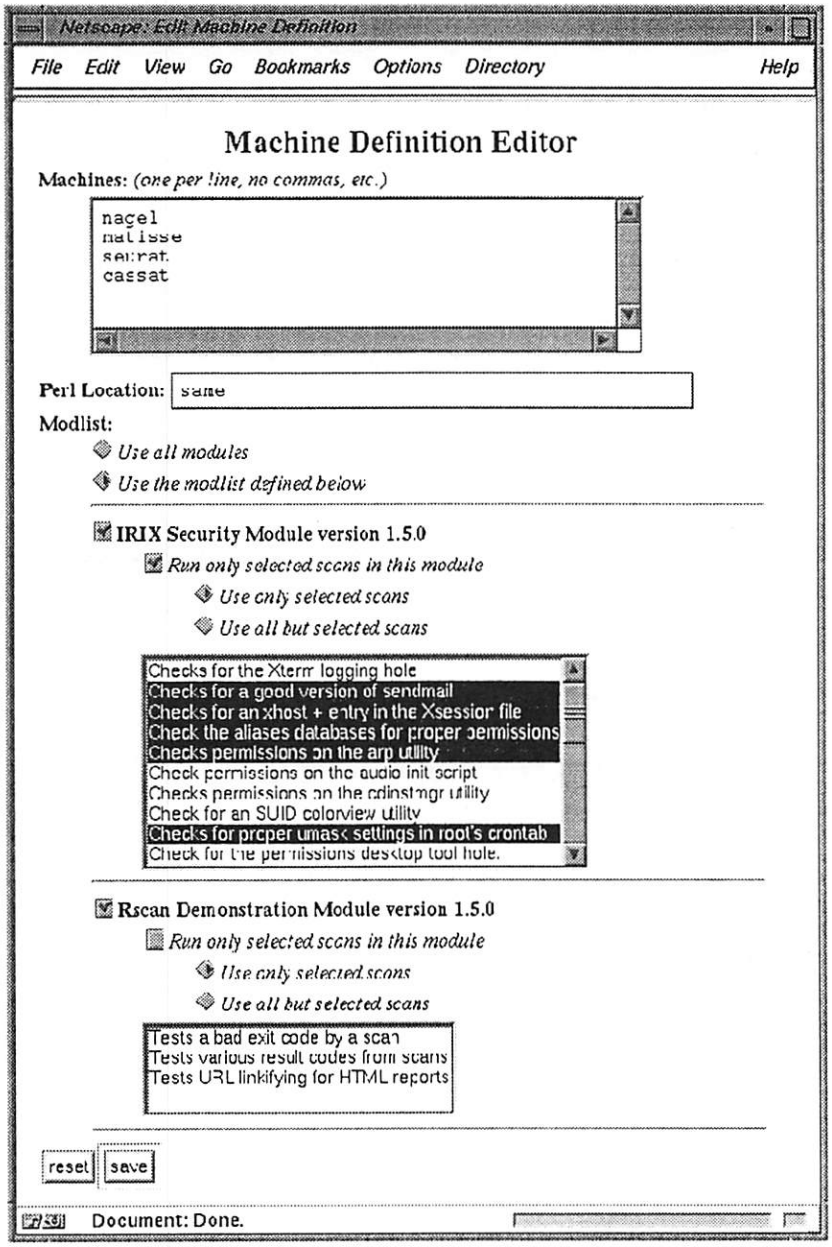


Figure 5: GUI Machine Definition Editor

when using them on the command line, since the '[', ']' and ':' characters are treated as meta-characters by some shells. They should *not* be protected when used in the configuration files.

A Sample Run

As Rscan runs, it gives status messages about its activities to standard out. Such messages may notify the user that Rscan is copying the scanner and modules to the remote machine, or running its initialization scripts, etc. Figure 6 shows a sample run of Rscan in text mode and Figure 7 shows the same run output when run from within the GUI.

```
Multi-Platform Interrogation and Reporting

Version 1.5.0

Initializing
+------------------------------------------------------------------------+

Copying scanner to cassat
Initiating a remote scan on cassat
Initializing

Rscan 1.5.0 starting scan on cassat
Date: Monday, June 5 1995 at 10:55:20 am

 Test                                                          Condition
-------------------------------------------------------------------------

Rscan Demonstration Module version 1.5.0
 Bad exit code ..................................................... [ PASS ]
 Looking at API Variables .......................................... [ PASS ]
 Exit with failed .................................................. [ FAIL ] *
 Exit with info .................................................... [ INFO ]
 Exit with passed .................................................. [ PASS ]
  .
  . [ data deleted for brevity ]
  .
 Exit with nothing ................................................. [ ERR  ]
 Exit with warning ................................................. [ WARN ] *
 Exit with nothing ................................................. [ ERR  ]
 Test URL linkify .................................................. [ PASS ]
 Exiting seurat

+------------------------------------------------------------------------+

Please read the file
  ./reports/rscan.network-05.05.95-10:55:17.html
for a full scan report.

 It is in HTML format and should be read with a tool like
 Netscape Navigator, NCSA Mosaic or Lynx.
```

Figure 6: Sample text mode run output

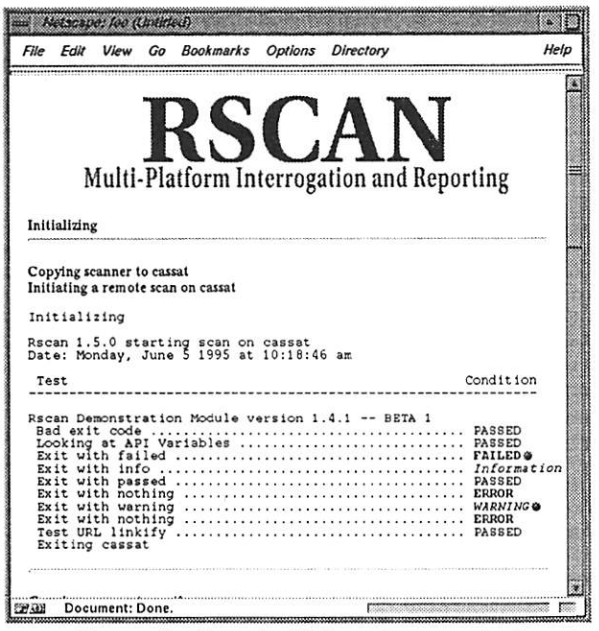


Figure 7: Sample GUI mode run output

If the report is written in HTML, then each scan's name is a link to the part of the report that more fully describes that scan's findings, and all text that resembles a URL is converted to a "real" URL. Figure 8 shows an ASCII report generated by the above run and Figure 9 shows an HTML report generated by the same run.

```
                    Rscan 1.5.0 Network Report
+------------------------------------------------------------------------+

Report for cassat

The date is:            Monday, June 5 1995 at 10:57:16 am
Rscan Version:          1.5.0
Perl Version:           5.001
Scans:                  Rscan Demonstration Module version 1.5.0
Modlist:                demo
Machine name:           cassat (cassat.VIS.ColoState.EDU)
Operating System:       IRIX 5.3
OS Patch Level:         11091812
CPU Type:               IP22 mips
System ID:              1762128434

 Test                                                          Condition
-------------------------------------------------------------------------

Rscan Demonstration Module
  Bad exit code ..................................................... [ PASS ]
  Looking at API Variables .......................................... [ PASS ]
  Exit with failed .................................................. [ FAIL ] *
  Exit with info .................................................... [ INFO ]
  Exit with passed .................................................. [ PASS ]
  Exit with nothing ................................................. [ ERR  ]
  Exit with warning ................................................. [ WARN ] *
  Exit with nothing ................................................. [ ERR  ]
  Test URL linkify .................................................. [ PASS ]
-------------------------------------------------------------------------

Test Data for Rscan Demonstration Module

 Test: Bad exit code
     This scan will exit with code 42, as opposed to 0.
     Test Results: PASSED

      WARNING: scan "demo/badexit.pl"
        exited with status = 42.  There may have
        been an error in it's run. PID was 1069.

  .
  . [ data deleted for brevity ]
  .
 Test: Test URL linkify
     This is a url:
       ftp://ftp.vis.colostate.edu/pub/rscan
     and so is this:
       http://www.vis.colostate.edu/rscan
     and so is this:
       http://www.vis.colostate.edu/info/staff/nate
     Test Results: PASSED

+------------------------------------------------------------------------+
```

Figure 8: Sample ASCII report

Programming Rscan

Rscan can be used to provide integrity checks for a site or to automate any number of complicated or mundane tasks. This section describes how to write custom Rscan modules.

The Rscan API

Rscan has a set of functions for writing to the screen, reports, and performing other common tasks. These functions should be used if applicable, and writers of modules are invited *not* to rewrite them, since it makes life hard for the writer of the module when a new version of Rscan comes out with changes.

API Functions

&pcheck(LIST) The string formed by the joining of *LIST* is written to the check list. This function should be used to signal the current test that is being run. There can be more than one `&pcheck` call in a scan, but each one should have a corresponding call to one of the result functions (`&passed`, `&failed`, `&warning`, and `&info`) before the next call to `&pcheck` is made. If there are two calls to `&pcheck` without a result function call in between or if a scan exits with an unresolved call to `&pcheck`, Rscan catches this, reports it and inserts an error condition (with `&error`).

&passed The last test should be marked as passing.

&failed The last test should be marked as failing.

&warn The last test should be marked as a warning.

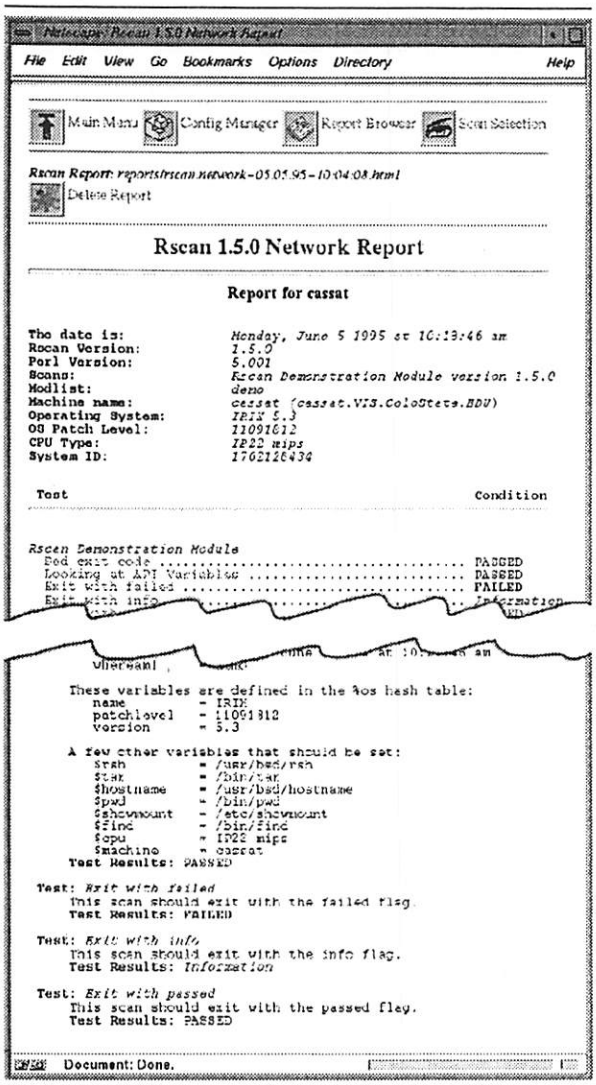


Figure 9: Sample HTML report

&info The last test should be marked as informational.

&report(LIST) The string formed by the joining of *LIST* is written to the report file. This function should be used to write messages to the report file. There is no need to write different things for an ASCII and an HTML report, since Rscan takes care of formatting automatically. If reports are being written in HTML, then any text printed using this function that resembles a URL is converted to a hypertext reference.

&screen(LIST) The string formed by the joining of *LIST* is written to the screen. This function should be used to write messages to the screen about what is happening. It is not recommended for use in scans, but primarily in the `init` files for a module.

&center(LIST) Returns the joining of *LIST* centered on a 65 character wide page.

&nprint(space,text) Returns *text* left justified in a field *space* characters wide, padded with spaces. This function is good for making formatted tables in reports, etc.

&header(headername,Value1,...,ValueN) Places a header field named *headername* in the report's header section for the current machine. The values *Value1* ... *ValueN* are placed to its right such that they appear in the report's header as follows:

```
headername: Value1
            Value2
            ValueN
```

There is no limit to how many values can be specified. The text for the *headername* is printed using `&nprint` in a 25 space wide field, and successive text is indented 25 spaces. Like the `&screen` function, this is primarily meant to be used in the `init` files for a module.

&rawheader(LIST) The text formed by the joining of *LIST* printed *without any additional formatting* into the header section of the report. This could be used to place some text centered in the header, etc. Again, this function should only be used in the `init` files for a module.

API Variables

Most API variables are collected in the `%api` associative array. Variables of particular interest are:

$api{'whereami'} This is the directory that the currently running script is located in. This seems pretty lame, but the script files move around depending on where the temp directory is located, etc. This variable should be used, for instance, when opening a database associated with a scan, and when doing other path oriented operations.

$api{'scanmode'} This is either set to `local` or `remote` depending on whether the scanner is running locally or remotely. The important difference between the two is that in local mode, scans are run *in place* and when run in remote mode they are run from within a temporary location and are deleted after the scanning run has been completed.

$api{'scannerdir'} This is set to the base directory that the scanner running in. It is set to something like `/tmp/scanner` or `/var/tmp/scanner`, depending on the `-temp` command line option and `tempdir` configuration file option.

$api{'outformat'} This is either set to `ascii` or `html` depending on whether the reports are being written as plain ASCII or HTML files. Again, because Rscan automatically formats output, writing different text for ASCII and HTML reports is discouraged, since Rscan will

automatically format the output accordingly (this variable is here in case users want to do this anyway).

$api{'perlversion'} This is set to the version number of `perl` that is running the scan.

$api{'now'} This is set to the integer time (the number of seconds since January 1, 1970) that `perl` was started at.

$api{'time'} This is set to the human-readable time that looks something like "Friday, January 1, 1970 at 05:22:44 pm"

There is also a `%os` associative array that contains information about the operating system of the machine currently being scanned. Here are the `%os` variables, along with some other variables that pertain to the operating system and the machine in general.

$os{'name'} The name of the operating system. Something like `IRIX` or `SunOS`, as returned by `/bin/uname`.

$os{'version'} The version of the operating system. Something like `4.0.5` or `6.0.1`, as returned by `/bin/uname -r`.

$os{'patchlevel'} The operating system's patch level, as defined by `/bin/uname -v`.

$machine The short name of the machine, as defined by `/bin/uname -n`.

$cpu The machine's CPU type, as defined by `/bin/uname -m`.

$rsh The location of the `rsh` utility.

$tar The location of the `tar` utility.

$tar_create The command line options necessary to have `tar` create an archive sending its output to standard out to be used as input to a pipe.

$tar_extract The command line options necessary to have `tar` extract an archive non-verbosely.

$gzip The location of the `gzip` utility.

$gzip_extract Command line options to give `gzip` to extract an archive to standard out (so it can be piped to `tar`).

$remove How to delete files and directories. Usually `/bin/rm -rf`.

Porting Rscan to a New OS

In general, Rscan should run on any UNIX that has either `perl` version 4 or 5 installed on it. There is one operating system specific file that has to be created for each new operating system that Rscan will be run on. This file is called the "pathconfig" file for the operating system, and is located in the `scanner/pathconfig` directory and must be named the same as the output of `/bin/uname` on that operating system.

This file should contain definitions for how to obtain the variables in the `%os` associative array (`name`, `version`, etc.) and the absolute paths to several necessary programs. New pathconfig files should be modeled after the others stored in the `pathconfig` directory. The really important options are the definition of where `rsh` and `tar` are located and how to extract a `tar` archive. If Rscan will be run in GUI mode under perl version 4 on the new operating system, then the pathconfig file must also include definitions for the `$AF_INET`, `$PF_INET` and `$SOCK_STREAM` variables associated with sockets (in perl 5, these variables are available from function calls, but in perl 4 they must be predefined). The best way to get a feel for what these scripts do is to read through a few of the other pathconfig files and give it a shot... the worst thing that can happen is that Rscan will not run the scans properly!

Module Structure

Modules are organized by short name in the "`scanner/modules`" directory under the main Rscan directory. Figure 10 illustrates the structure of a typical module.

```
module/                          Main directory for the module
      init                       Generic initialization file
      fullname                   Module's "Full Name"
      version                    Module's version number
      desc                       descriptions of all *.pl files
                                 in this directory
      *.pl files                 individual scan files
      OSNAME1/
            init                 Initialization data for OSNAME1
            VERSION1/
                  desc           descriptions of all *.pl files
                  *.pl files     scans for OSNAME1 VERSION1
            VERSION2/
                  desc           descriptions of all *.pl files
                  *.pl files     scans for OSNAME1 VERSION2
            VERSIONn/
                  desc           descriptions of all *.pl files
                  *.pl files     scans for OSNAME1 VERSIONn
            common/
                  desc           descriptions of all *.pl files
                  *.pl files     scans for all versions of OSNAME1
      OSNAME2/
            init                 Initialization data for OSNAME2
            VERSION1/
                  desc           descriptions of all *.pl files
                  *.pl files     scans for OSNAME2 VERSION1
            VERSION2/
                  desc           descriptions of all *.pl files
                  *.pl files     scans for OSNAME2 VERSION2
            VERSIONn/
                  desc           descriptions of all *.pl files
                  *.pl files     scans for OSNAME2 VERSIONn
            common/
                  desc           descriptions of all *.pl files
                  *.pl files     scans for all versions of OSNAME2
      OSNAMEn/
            init                 Initialization data for OSNAMEn
            VERSION1/
                  desc           descriptions of all *.pl files
                  *.pl files     scans for OSNAMEn VERSION1
            VERSION2/
                  desc           descriptions of all *.pl files
                  *.pl files     scans for OSNAMEn VERSION2
            VERSIONn/
                  desc           descriptions of all *.pl files
                  *.pl files     scans for OSNAMEn VERSIONn
            common/
                  desc           descriptions of all *.pl files
                  *.pl files     scans for all versions of OSNAMEn
```

Figure 10: Example module directory structure

Module initialization scripts are called "`init`" and (if present) are located in the base directory for the module, and in each subdirectory therein named for an operating system (e.g., "`IRIX`," "`SunOS`," etc.). These scripts (if present) are run when a module is executed on a remote machine. First the `init` script in the main module directory is run, and then the `init` script in the directory named for the operating system of the machine that the scanner is running on is run. A file named "`fullname`" located in the base directory for the module contains a one line "long name" for a module. A file called "`version`" in the same directory contains the version number of the module.

In each directory that contains actual scanner scripts (including the main module directory), there

is a file called "desc" which contains one line descriptions for what each scan in that directory does. Rscan uses these files when listing the modules that are installed. They make it easier to understand what a particular scan will be doing when its run. The format for the desc files is as follows:

```
scanner_script.pl
one-line desc. of "scanner_script.pl"
```

Blank lines and lines beginning with the "#" character are ignored.

Running Order For Files

When Rscan starts its run on a machine, files are always executed in the following order (if they are present):

1. The generic initialization file for the module is run. This file is located in the main module directory and called `init`.
2. The OS-specific initialization file for the module is run. This file is located in the directory named for the current operating system under the main module directory, and is also called `init`.
3. `.pl` files in the main module directory are run in alphabetic order.
4. `.pl` files in the *osname* `/common` directory under the main module directory are run in alphabetic order.
5. `.pl` files in the *osname* / *osversion* directory under the main module directory are run in alphabetic order.

As discussed previously, the modlist determines the exact set of `.pl` files which will be run on any particular machine, but they are always run in alphabetic order.

Tips For Writing Effective Modules

Writing modules for Rscan is very easy, but there are a few tips that will make writing them much easier and make running them go much smoother.

Instead of having many many small scripts, try to collect some of the scripts into one larger script. This is especially true if one script must be run before another script can do its job – which may not happen if the user decides not to run one of the scripts by not including it in a modlist.

Remember that since each script in each module is run in its own process, any variables that are set up in one script will *not* be available to any other script in the module. If there is a need for some variables to be set up for a particular operating system or for every operating system that a module will be run on, they should be set in one of the `init` scripts.

When there is a task that needs to be accomplished on all the different operating systems that the module will run on, but needs to be done in a slightly different way on each system, have a separate script for each operating system (and possibly one for each version of each operating system), but name each script with *the same file name.* This way, if the user wants to accomplish that task on a set of machines, the user simply needs to include that script name in the modlist for those machines and the correct script for that operating system will be run, since there is no way (other than multiple machine definitions) to set a script to be run only under some particular version of an operating system. For instance, let's say that someone is writing a module to check the configuration of each machine on their network and one of the checks is to make sure that the correct sendmail configuration file is present on each machine. Since sendmail configuration files can differ drastically from one operating system to another, the user should write a separate script for each operating system (and, if necessary, a separate one for each version of an operating system) and call each one, for instance, `checksendmail.pl`. Now, when the user wants to check the sendmail configuration files on all their machines they simply need to use a modlist like "`mymodule[checksendmail]`" and the proper sendmail configuration file check script will be run on each machine.

A Programming Example

Rscan modules can be trivially simple or extremely complicated. This is a step-by-step description of what is required to write a module that incorporates all of Rscan's major features.

Let's say that an administrator wants to run Tripwire on each machine on their network. The first thing to do is compile a copy of Tripwire for each architecture that the module will be run on. If there are different options needed to run on different versions of one or more of the operating systems in question, the administrator must build a binary for each of them.

Next, write a perl script for each of the binaries that were created above. These may all be identical unless there's some strange things that need to be done on certain operating systems. They could be as simple as just starting the process and with the right command line options for each architecture and using the `&report()` function to relay that information back to the server, like this:

```
open(PROC, "mytool -arg1 -arg2 |");
while (<PROC>) {
  &report($_);
}
close(PROC);
```

The script could also move around the Tripwire databases as needed before running Tripwire itself, make backups of the databases, etc. The scripts should all be named the same for reasons explained in the previous section.

Now that the wrapper scripts have been written, organize the module in the way shown in Figure 10. The module should now be ready to run. It's that simple. The only thing that needs to be kept track of now is the name of the module and the name of the scripts in the module. This process can easily be repeated for other common tools like Crack, TAMU, or other custom scripts or programs that are in use in an organization.

Future Enhancements

The only real enhancement planned for Rscan is the use of secure communications channels between the server and the remote machine and between the HTTP servers and the WWW browser. For securing communications between the server and the remote machine, Netscape's Secure Socket Library is a possible solution, but the SSH (Secure Shell) Remote Login Program from Tatu Ylönen (*ylo@cs.hut.fi*) is probably better, since the best solution for securing this part of Rscan is to implement a secure version of `rsh` and use that, since it would make Rscan secure with no modifications at all. Securing the communication between the Rscan HTTP servers and the WWW browser would almost certainly require the use of the Netscape SSL library since Netscape Navigator is currently the only security-enabled WWW browser. Unfortunately, the use of any kind of secure communications usually conflicts with ITAR export regulations (because most encryption schemes are classified as weapons by the US Government and are therefore subject to the same export regulations as nuclear weapons), so a secure Rscan would probably not be exportable.

Suggestions for future enhancements (and bug fixes) are very welcome, and reasonable requests for new features are usually implemented. Requests should be sent directly to the author.

Availability

Rscan is available via anonymous ftp to `ftp://ftp.vis.colostate.edu/pub/rscan`. The file `rscan.tar.gz` is always a link to the most recent copy of Rscan. The Rscan WWW homepage is located at `http://www.vis.colostate.edu/rscan`.

Beta releases of "in the works" copies of Rscan are placed in the `/pub/rscan/beta` directory. The archives in that directory should be considered pre-release, and are not recommended for use in a "production" environment. They are only there for evaluation and bug reports by those people who are kind enough to test the software.

There is also a mailing list that receives announcements about new versions of Rscan and about new modules. The list name is `rscan-announce@vis.colostate.edu` and may be subscribed to by sending mail to `majordomo@vis.colostate.edu` and including the text "`subscribe rscan-announce`" on a line by itself in the message body. The list is moderated by the author, and is extremely low traffic.

Rscan is completely free, but out of curiosity the author requests that users send mail regarding their use of Rscan. The author also requests that if a user hacks up the code and redistributes it they don't take full credit for writing it from scratch.

Biography

Nathaniel (Nate) Sammons is currently a Computer Science student at Colorado State University where he works for Academic Computing and Networking Services in the Computer Visualization Laboratory. He has been the system administrator there for the past two years. Nate can be reached electronically at *nate@colostate.edu*. His WWW homepage is `http://www.vis.colostate.edu/info/staff/nate`.

References

[1] Dan Farmer and Eugene H. Spafford. The COPS security checker system. *USENIX Conference Proceedings*, Pages 165-170, Anaheim, CA, Summer 1990.

[2] Larry Wall and Randal L. Schwartz. *Programming Perl*. O'Reilly & Associates, Inc. Sebastopol, CA, 1991. ISBN 0-937175-64-1.

[3] David R. Safford, Douglas Lee Schales, and David K. Hess. The TAMU Security Package: An Ongoing Response to Internet Intruders in an Academic Environment. *Proceedings of the Fourth Usenix UNIX Security Symposium*, Pages 91-118, Santa Clara, CA, October 1993.

[4] T. Berners-Lee, R. T. Fielding, and H. Frystyk Nielsen. Hypertext Transfer Protocol – HTTP/1.0, Third Edition. *IETF HTTP Working Group*. March 8, 1995.

Exu – A System for Secure Delegation of Authority on an Insecure Network

Karl Ramm – Massachusetts Institute of Technology
Michael Grubb – Duke University

ABSTRACT

Administration of a large and complex system poses several problems: Usually, some tasks must be delegated due to lack of qualified or trusted staff, and some tasks must be automated. In many cases, some parts of the task might need special credentials, such as Kerberos tickets or AFS tokens, that may not necessarily be easily available to the person executing the task. The problem is that most systems divide users into two groups: haves and have nots, and provide no mechanism for finer-grained access control. In addition, the tasks executed must be carefully recorded for possible later auditing. Earlier solutions, such as the setuid bit, Moira, ADM, and sysctl, can be used to accomplish this, either in a limited or dangerous (in the case of setuid) fashion. Exu proposes to solve the problem via secure, authenticated connection to a server with full authentication that can cause things to happen in real time.

Problem

In a system with a very large number of users and a very small number of administrators, "superuser" access is of necessity hoarded. If such a system is critical to the operation of the enterprise (in this case, a university) the need to be careful is even more crucial. On the other hand, work-a-day system administration tasks such as changing forgetful users' passwords for them, new user account creation, and the maintenance of mailing lists are nice to farm out when you spend your days putting out fires and placating angry users. Additionally, certain tasks often end up being performed by hand or with ugly kludges involving frightening authentication structures, because authorization tends to be an all-or-nothing proposition, dividing users into haves and have-nots.

Some systems, such as TransArc AFS [AFS], partially address this problem with access control lists (ACLs), allowing several different entities to have varying amounts of access to different services; there's an ACL for "superuser" filesystem access, an ACL for doing fileserver maintenance, and another ACL for altering the kaserver (Kerberos) database. This begins to break down when you want a certain person to be able to do certain fileserver operations such as moving files between certain disks, or back them up, without giving them carte blanche for the rest of the system.

In the situation at Duke University's Office of Information Technology, with three system administrators, fourteen server machines, over one hundred workstations, and over nineteen thousand user accounts, there was very little time to develop the infrastructure for the "right solution" to the problem, and things needed to be implemented very quickly. Thus, any labor saving device that was employed had to be very flexible and quickly extensible.

This situation led to the development of Exu, a service designed to delegate authority in a controlled fashion and smoothly automate system administration tasks requiring authentication to multiple services. A third, emerging motivation is the elimination of IP-address based authentication to counter emerging security threats on the Internet. [IPSecurity] Exu rejects the traditional all-or-nothing type of security, and the not quite as traditional ACL based security, for procedural security, where a completely configurable piece of code determines the level of authorization.

Other Approaches

Setuid Bit

One of UNIX's more interesting innovations was a per-file bit that allowed executables to assume the authentication of their owner. This is used by utilities such as the local-file password-changing program to provide controlled write access to security-critical files. On a typically less than totally secure network, the setuid bit leads to a variety of problems: in cases involving remote file systems, one's trust of the file server must be contemplated, given that it probably could be easily spoofed, and if the local machine has sufficient privileges, the security of all the machines that trust the file server (which probably includes the file server) is predicated on the local machine's security. Also, applications of the setuid bit do nothing to address the problem of monitoring. Given that most workstations are interconnected via bus networks such as Ethernet, encryption and a method of authentication that does not require secret information transmitted in the clear is

necessary. Such a system is MIT's Kerberos [Kerberos], which uses shared secrets to authenticate connections and securely transport randomly-generated session keys.

MIT Athena SMS

The Athena Service Management System [Moira] is fundamentally a database driver: It uses a commercial database (RTI Ingres) to store information concerning workstations, server configurations, users, and printers, with an input driver that handles access control, information queries and updates, as well as an output driver that actually updates the configurations and password files every night. Authentication and wire security are handled via Kerberos. Although there are many interesting applications that talk to it, the server itself doesn't do anything more than maintain the database. The program itself is not easily extensible; if you want to control a new set of files, you need to write the output stages and the input stages in C.

CMU ADM

The CMU ADMinistration server [FlexAdmin] is on the opposite end of the spectrum. It essentially consists of a Scheme interpreter that can run two modes: user mode and privileged mode. The privileged mode, which is normally restricted to system administrators, has access to a variety of primitives, in the canonical example being AFS server control operations, and can define functions which can execute these dangerous primitives *even in user mode*. The catch is, of course, that the interpreter in user mode cannot modify these functions. The idea is that these functions are written in a manner so as to guarantee their security – that is, check who's calling them before they perform the operation. Authentication and wire security is, as usual, provided by Kerberos. Unfortunately for our situation, an incomplete implementation and lack of provision for a database package hampered extensibility and usefulness, although it does have procedural access control.

Sysctl

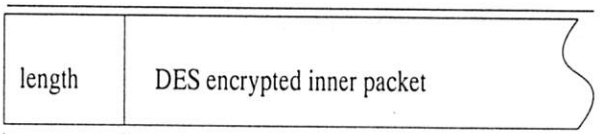
The sysctl package from IBM's Project Agora has a similar idea; however, it uses Tcl [Embeddable], a scripting language that takes syntactical concepts from the Bourne shell, C, and LISP, giving each procedure either one of three security levels: unauthenticated (for people with no Kerberos authentication), authenticated (for people with), and trusted (for people who are authenticated and on the list of trusted users), or a specific ACL. Although it should be possible to do procedural security in this context, it is not directly supported by the paradigm. Sysctl handles authentication and wire security via Kerberos.

Exu

Exu (pronounce *ee-shoo*) consists of a single-threaded server process, running out of *inetd*(8). A server will run for each client program in order to simplify security concerns at the expense of a possible performance bottleneck. Exu is extensible via Tcl. Tcl is also easily extensible; there are a variety of packages that add database functionality, access to most of the UNIX system call interface [XTcl], and even an extension called expect [Automate], which permits extensive control and management of subprocesses, even those that think they're interactive. The Exu implementation also adds commands to manage various AFS entities. However, the most important extension to Exu is SafeTcl [MailEnabled], which consists of a pair of Tcl interpreters, the restricted interpreter and the unrestricted interpreter. The restricted interpreter has all possible "dangerous" commands (file manipulation, program execution) removed, and the unrestricted interpreter has all the interesting extensions (expect, extended Tcl) added, as well as primitives for allowing *unrestricted* interpreter procedures to be executed in the *restricted* interpreter. This produces an effect similar to the privileged/unprivileged modes in ADM, above, and allows the same sort of procedural access control, with the extensibility allowed by the TCL in sysctl. These procedures are organized into groupings called libraries, which can be loaded with the loadlibrary command. Exu, of course, handles authentication and wire security via Kerberos.

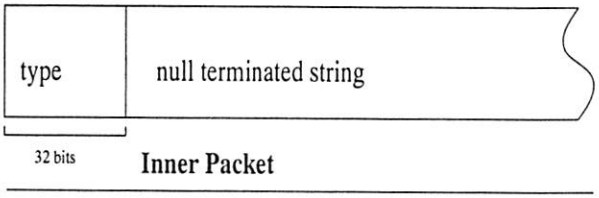
Protocol

The Exu protocol is conceptually very simple: The client creates a TCP connection to the server, exchanges Kerberos authentication information via the *krb_sendauth* and *krb_recvauth* library routines, and then the client and server send logical "packets" back and forth over it. Each packet consists of an "outer packet" and an "inner packet". The outer packet consists of a 32-bit network byte order integer which represents the length of the rest of the packet, followed by that number of bytes which represent the inner packet DES encrypted with the Kerberos session key.



Outer Packet

The inner packet consists of another 32-bit network byte order integer followed by a null-terminated string. The integer is the "message type" code, and the string is the message itself.



At the moment, the message type number is only used for Tcl return codes; any string sent to the server is presumed to be a Tcl command string. In the future, the client to server message type field may be used for things like client-side logging of information, and the other direction may be used for server output.

Client Library

The Exu client library is conceptually very simple, with only three functions: *exu_open* and *exu_close()* which open and close a connection to the Exu server, and *exu*, which sends a command to the Exu server for execution. At the moment, the syntax of these functions is such that they can be directly linked into a Tcl interpreter, although a future version will export C bindings as well as Tcl bindings.

Example – kasexam

Here is an example of some code for a simple Exu client using the client Tcl bindings. Note that the switch between client execution and server execution is very clean looking, because they're both Tcl code.

Sample Code – kasexam.tcl

```
#!/path/to/exush

# Open Exu server
set server [exu_open blake7.duke.edu]

# exu_open returns name of
# server principal
puts "Remote: $server"

exu {
  loadlibrary kas
}
# do a remote loadlibrary command
# (loadlibrary is a renamed
# SafeTcl_loadlibrary)

puts [exu "examine [lindex $argv 0]"]
# retrieves and prints out ugly pile
# of information

exu_close
# close the server connection
```

Sample Run

Here is a sample run of the above code:

```
% kasexam hiro
Remote: exu blake7 ACPUB.DUKE.EDU
1 normal -1 785454095 admin {} \
776011185 90000 0
%
```

Server code

Here is an extract from the kas "library" of the functions that were executed on the server in the above example:

```
# use a regular expression to turn
# name.instance@REALM to
# {name instance REALM}
proc parse_princ {name} {
  regexp  {([^.@]*)(\.([^.@]*)|)\
    (@(.*)|)} $name foo m1 m2 m3 \
    m4 m5 m6
  set l {}
  lappend l $m1
  lappend l $m3
  lappend l $m5
  return $l
}

# retrieve incidental information
# from Kerberos Authentication
# Service database; make sure that
# the caller is karl@ACPUB.DUKE.EDU
proc examine {user} {
  set p [parse_princ $user]
  if {[exu_client_principal] ==
       {karl {} ACPUB.DUKE.EDU}} {
    return [kas_examine [lindex $p 0] \
      [lindex $p 1]]
  }
}
declareharmless examine
```

Note the *exu_client_principal* Tcl command, which returns to the calling function the Kerberos principal that the client authenticated as. Also note the *declareharmless* command, which is used to make a function available to the restricted interpreter.

Choice of Tcl

Tcl has many advantages in this sort of application: it is well supported and there is a lot of code floating around for it because for anyone reasonably proficient in C or Bourne shell programming, Tcl code can be generated very rapidly. The existence of extensions such as Expect and SecureTcl also makes it very useful for this sort of application.

Unfortunately, many of the features of Tcl which make it easy to program mask potential flaws in its design, particularly its quoting. Any time you use the `eval` command, unless you are being very careful about where you got your input from, you run the risk of executing arbitrary code. An example (and this *is* somewhat contrived) follows:

```
proc runcommand {a} {
    # where a is actually an argument
    # to another function, doit
    eval doit $a
}
proc doit {args} {
    foreach i $args {
        puts $i
    }
}
```

If you send it

```
runcommand {a b c}
```

where {a b c} is the quoted string "a b c", you get

```
a
b
c
```

instead of

```
a b c
```

as you might expect. There are ways to program around this, but it's merely annoying, right? Wrong. It's dangerous, because if you run

```
runcommand {foo [puts gotcha]}
```

you don't get

```
foo
[puts
gotcha]
```

as you might now expect, but

```
gotcha
foo
```

Because the Tcl interpreter *in the context of the doit runcommand procedure* executed the `puts gotcha` and substituted the result (an empty string). This could have suboptimal results if you happen to be running as root on a sealed machine at the time. As mentioned before, there are ways to program around this, but it often requires some thought.

Another large drawback of Tcl is the fact that it does not support (at the moment) standard dynamic loading, which means that when you incorporate all your interesting extensions into your server interpreter, it's not so lightweight anymore.

If there were a nice clean language, with somewhat stronger typing and somewhat less abstruse quoting conventions and a standard mechanism for dynamic loading (and a reasonable security model), then it would probably be a better choice for this sort of thing.

Applications

homepage

One of the flagship uses of Exu at Duke is a service called *homepage*. Homepage allows users to remotely add, change, and remove their own entries in the campus web server's listing of web home pages, without having login access to the web server or access to its filesystem. The Exu server software has proven to be very reliable under Solaris and Ultrix. The homepage client side is also implemented in Tcl and offers to set up the user with a blank template home page if they don't have one yet.

Future Thoughts

In addition to further implementation of AFS Tcl extensions to the Exu server library, Exu is well-suited to delegated management of portions of large databases, such as are used in site-wide account administration and large-scale mailing list management. There also needs to be an unauthenticated mode (probably without encryption), and a library for handling ACLs. Another interesting application might be for non-privileged users maintaining software in replicated AFS volumes. In some environments, where privacy is a concern and web page access statistics might be considered privileged information, users could use Exu to retrieve usage statistics on their pages.

User database

Having a master user database from which all password tables, NIS maps, NIS+ maps, Hesiod tables, etc. are updated and generated from is an old idea. We envision a database with the person's account name, uid number, real name, student/employee id number (SSN), home directory, mailbox location and special access privileges. This can be implemented trivially with any of the several Tcl database extensions. Alterations could be pushed out nightly, or be instantly updated depending on the type and priority of the update, and on the service being updated. Exu could thus handle all phases of user account creation and maintenance.

If user accounts are managed entirely by Exu, it becomes simple to leverage that into a pretty Tk-based user management tool. Such a tool simplifies administrative tasks for non-technical staff, and is very good for impressing the natives.

This approach can also provide working *chfn*(1) and *chsh*(1) commands to allow users to alter their own full name and login shell entries in the master database which otherwise is problematic on distributed or otherwise heterogeneous systems (such as Ultrix Hesiod).

Mailing List Management

Exu could also be used to help manage a mailing list server. List owners could use a simple line-oriented menu interface or a Tk-based GUI to perform administrative functions such as altering descriptions, changing the list password, and insertion and deletion of list members. Mailing list subscribers with Kerberos principals in the local realm would similarly be able to use an Exu client to

browse available lists and subscribe and unsubscribe. Since configurational and administrative tasks are notoriously CPU-intensive in some widely-used mailing list management server software, Exu could easily be a performance boost to a mailing list service if used in this way.

Software Availability

The current Exu distribution is available by anonymous ftp from ftp.duke.edu in */pub/exu*.

Conclusions

Exu should be a highly useful tool to any sufficiently overworked set of administrators. The design is optimized for flexibility, not performance; it strives for maximizing the capability/complexity ratio. The approach is new; long term performance in a production environment is yet to be measured.

Author Information

Karl Ramm is a systems programmer for MIT Information Systems, where he helps keep Athena from falling over. He can be reached via U.S. Mail at Room E40-342B, Massachusetts Institute of Technology; 1 Amherst St.; Cambridge MA 02139, while he can be reached electronically at kcr@mit.edu. He has written five different mail reading programs and has never written a windowing system.

Michael Grubb is a systems programmer for Duke University's Office of Information Technology. He is also a licensed attorney. He can be reached by post at Box 90132, Durham, NC 27708-0132 USA or by email at mg@ac.duke.edu.

References

[IPSecurity] Morris, R.T., *A Weakness in the 4.2BSD Unix TCP/IP Software*, Computing Science Technical Report No. 117, AT&T Bell Laboritories, Murray Hill, New Jersey, 1985.

[Kerberos] Steiner, Neuman, Schiller, "Kerberos: An Authentication Service for Open Network Systems", *USENIX Technical Conference*, Dallas, Texas, Winter 1988.

[Moira] Rosenstein, Geer, Levine, "The Athena Service Management System", *USENIX Technical Conference*, Dallas, Texas, Winter 1988.

[FlexAdmin] VanRyzin, "Flexible Administration for AFS", *AFS Users Group*, Fall, 1991.

[SysCtl] DeSimone, Lombardi, "Sysctl: A Distributed System Control Package", *LISA VII*, Monterey, California, 1993.

[Embeddable] Osterhout, "Tcl: An Embeddable Command Language", *USENIX Tehnical Conference*, Washington, D.C., Winter 1990.

[XTcl] Lehenbauer, Diekahns, "Extended Tcl", ftp://ftp.neosoft.com/languages/tcl/distrib/tclX7.4a-b2.tar.gz

[Automate] Libes, Don, "Using expect to Automate System Administration Tasks", *LISA IV*, Colorado Springs, Colorado, 1990.

[AFS] Howard, John, "An Overview of the Andrew File System", *USENIX Technical Conference*, Dallas, Texas, Winter 1988.

[MailEnabled] Borenstein, Rose, "MIME Extensions for Mail-Enabled Applications: application/Safe-Tcl and multipart/enabled-mail", unpublished draft distributed with SafeTcl, ftp://ftp.ics.uci.edu/pub/mrose/safe-tcl/safe-tcl-1.2.tar.Z

Administering Very High Volume Internet Services

Dan Mosedale, William Foss, and Rob McCool – Netscape Communications

ABSTRACT

Providing WWW or FTP service on a small scale is already a well-solved problem. Scaling this to work at a site that accepts millions of connections per day, however, can easily push multiple machines and networks to the bleeding edge. In this paper, we give concrete configuration techniques that have helped us get the best possible performance out of server resources. Our analysis is mostly centered on WWW service, but much of the information applies equally well to FTP service. Additionally we discuss some of the tools that we use for day-to-day management.

We don't have a lot of specific statistics about exactly how much each configuration change helped us. Rather, this paper represents our many iterations through the "watch the load increase; see various failures; fix what's broken" loop. The intent is to help the reader configure a high-performance, manageable server from the start, and then to supply ideas about what to look for when it becomes overloaded.

Our Site

Netscape Communications runs what we believe to be one of the highest volume web services on the Internet. Our machines currently take a total of between six and eight million HTTP hits per day, and this number continues to grow. Furthermore, we make the Netscape Navigator, our web browser, available for downloading via FTP and HTTP[1].

The web site[2] contains online documentation for the Netscape Navigator, sales and marketing information about our entire product line, many general interest pages including various directory services, as well as home pages for Netscape employees. All of the machines run the Netscape Server, but most of the strategies in this paper should apply to other HTTP and even FTP servers also.

At various times, we have tried out various configurations of machines running the given operating systems; Figure 1 shows a list.

Each of our WWW servers has an identical content tree uploaded to it (more on this later). Before the Netscape Navigator was released to the Internet community for the first time, we thought about the web pages we intended to serve and debated how we would spread the load across multiple machines when that became necessary. Because of problems reported using DNS round-robin techniques[3], we chose to instead implement a randomization scheme inside of the Netscape Navigator itself. In short, when a copy of the Navigator is accessing home.mcom.com or home.netscape.com, it periodically queries the DNS for a hostname of the form homeX.netscape.com, where X is a random number between 1 and 16. Each of our web servers has a number of the homeX aliases pointing to it. Since this strategy is not something that will be available to most sites, we won't spend more time on it here.

Another scheme which we have looked into is nameserver-based load balancing. This scheme depends upon a nameserver periodically polling each content server at site to find out how loaded they are (though a less functional version could simply use static weighting). The nameserver then resolves the domain name to the IP address of the server which currently has the least load. This has the added benefit of not sending requests to a machine that is overloaded or unavailable, effectively creating a poor man's failover system. It can, however, leave a

Sun uniprocessor (60 Mhz) SPARCserver 20	Solaris 2.3 & 2.4
SGI 2-processor Challenge-L	IRIX 5.2
SGI Indy (150 Mhz)	IRIX 5.2 & 5.3
SGI Challenge-S (175 Mhz)	IRIX 5.3
HP 9000/816	HP-UX 9.04
90 Mhz Pentium PC	Windows NT 3.5
90 Mhz Pentium PC	BSD/OS 2.0

Figure 1: Tested hardware/software configurations

dead machine in the server pool for as long a duration as the DNS TTL. Two DNS load balancing schemes that we plan to investigate further are RFC 1794[4] and lbnamed[5].

Figures 2 and 3 summarize some of the performance statistics for the various servers on July 26, 1995 along with their assigned relative load. Our setup allows us to give each machine a fraction of the total traffic to our site measured in sixteenths of the total.

Note: In addition to assuming 1/16 of the WWW load, the 150 MHz Indy also wears the aliases www.netscape.com, home.netscape.com, and currently runs all CGI processes for the entire site. Total CGI load for this day accounted for 49,887 of its 1,548,859 HTTP accesses (more on this later).

The Tools

The traffic and content size of our web site grew very quickly, and we collected a number of tools to help us manage the machines and content. We used a number of existing programs, some of which we enhanced, and developed a few internally as well. We'll go over many of these here; the intent is to focus on tools that are directly useful in managing web servers and content. We will avoid discussing HTML authoring programs and utilities; for those who are interested, see http://home.netscape.com/home/how-to-create-web-services.html for pointers to many such tools.

Document Control: CVS

Since our content comes from several different sources within the company, we chose to use CVS[6] to manage the document tree. This has worked moderately well, but is not an ideal solution for our environment.

In some some ways, the creation of content resembles a mid-to-large programming environment. A document revision system became necessary to govern the creation of our web site content as multiple "contributing editors" added and deleted material from the content tree. CVS provided a reasonably easy method to retrieve older source or detailed logs of changes made to HTML dating back to the creation of the content tree.

One drawback of CVS is that many of the folks who design our content found it difficult to use and understand due to a lack of experience with UNIX. A cross-platform GUI-based tool would be especially well-suited to this market niche.

Host type	Load Fraction	Hits	Redirects	Server Errors	# unique URL's	Unique hosts	KB transferred
SGI 150Mhz R4400 Indy	1/16	1548859	68962	7253	4558	120712	12487021
SGI 175 Mhz R4400 Challenge S	6/16	2306930	47154	23	2722	249791	11574007
SGI 175 Mhz R4400 Challenge S	5/16	2059499	43441	43	2681	225055	10626111
BSD/OS 90 Mhz Pentium	4/16	1571804	31919	23	2351	192726	7936917

Figure 2: WWW Server activity for the period between 25/Jul/1995:23:58:04 and 26/Jul/1995:23:58:59

Host type	Load Fraction	Bytes/Hits	Bytes/Hits	Bytes/Hits	Bytes/Hits	Bytes/Hits
Host type SGI 150 Mhz R4400 Indy	1/16	430600/82	345132/61	227618/61	224849/60	236678/59
Host type SGI 175 Mhz R4400 Challenge S	6/16	613621/128	646112/119	656412/110	545699/108	520256/107
Host type SGI 175 Mhz R4400 Challenge S	5/16	466244/93	430870/88	358186/84	375964/84	531421/82
Host type BSD/OS 90 Mhz Pentium	4/16	375696/81	256143/77	417655/76	394878/72	298561/70

Figure 3: Five busiest minutes for the tested hosts

Content Push

Once we had multiple machines serving our WWW content, it became necessary to come up with a reasonable mechanism for getting copies of our master content tree to all of the servers outside our firewall. NCSA distributes their documents among server machines by keeping their content tree on the AFS distributed filesystem[3].

It seemed to us that another natural solution to this problem was rdist[7], a program specifically designed for keeping trees of files in sync with a master copy. However, we felt we couldn't use this unmodified, as its security depended entirely on a .rhosts file, which is a notoriously thin layer of protection. With the help of some other developers, we worked on incorporating SSL[8] into rdist in order to provide for encryption as well as better authentication of both ends. With SSL, we no longer need to rely on a client's IP address for its identity; cryptographic certificates provide that authentication instead.

In the development of our SSLified rdist, we decided that it would be a good idea to use the latest rdist from USC, in part because it has the option of using *rsh*(1) for its transport rather than *rcmd*(3). Because it doesn't use *rcmd*(3), it no longer needs to be setuid root, which is a real security win. One side effect of this is that we now have an SSLified version of *rsh*(1), which we use to copy log files from our servers back to our internal nets.

Monitoring

During the course of our server growth, we wrote and/or borrowed a number of tools for monitoring our web servers. These include a couple of tools to check response time, a log analyzer, and a program to page us if one of the servers goes down.

The tool to check response time is designed to be run from a machine external to the server being monitored. Every so often, it wakes up and sends a request to an HTTP server for a typical document, such as the home page. It measures the amount of time that it took from start to finish; that is, from before it calls connect() to after it gets a zero from read() indicating that the server has closed the connection. If you choose a relatively small document, this time can give you a good general indication of how long people are unnecessarily waiting for documents (since under ideal conditions a small document should come back nearly instantaneously). In our typical monitoring setup, we run monitor programs from remote, well-connected sites as well as from locally-networked machines. This allows us to see when problems are a result of network congestion as opposed to lossage on the server machines.

The logfile analyzer, which is now a standard part of the Netscape Communications and Commerce server products, provides information about the busiest hours or minutes of the day, and about how much data transfer client document caching has saved our site. The analyzer can be very helpful in determining which hours are peak hours and will require the most attention. Because of the high volume of traffic at our site, we designed it to process large log files quickly.

The program to page us when a server becomes unreachable is similar to our response time program. The difference is that when it finds that a server does not respond within a reasonable time frame for three consecutive tries, it sends an e-mail message to us along with a message to our alphanumeric-pager gateway to make sure we know that a server needs attention.

At times when we knew that we wouldn't be able to come in and reboot the server, we used a UNIX box with an RS232-controlled on/off switch to automatically hard boot any system not responding to three sequential GET requests. A small PC with two serial ports is enough to individually monitor 10 systems and provides recovery for most non- fatal system errors (e.g., most problems other than hardware failure or logfile partitions filling up).

Performance

Previous works [9, 10, 11] have explored HTTP performance and have come to the conclusion that HTTP in its current form and TCP are particularly ill-suited to one another when it comes to performance. The authors of some of these articles have suggested a number of ways to improve the situation via protocol changes. For the present, however, we are more interested in making do with what we have.

More practically speaking, most TCP stacks have never been abused in quite this way before, so it's not too surprising that they don't deal well with this level of load. The standard UNIX model of forking a new server each time a connection opens doesn't scale particularly well either, and the Netscape Server uses a process-pool model for just this reason.

Kernels

The problems that took the most time for us to solve involved the UNIX kernel. Not having sources for most platforms makes it something of a black box. We hope that sharing some hard-won insights in this area will prove especially useful to the reader.

TCP Tuning

There are several kernel parameters which one can tweak that will often improve the performance of a web or FTP server significantly.

The size of the listen queue corresponds to the maximum number of connections pending in the kernel. A connection is considered pending when it has not been fully established, or when it has been

established and is waiting for a process to do an *accept*(2). If the queue size is too small, clients will sometimes see "connection refused" or "connection timed out" messages. If it is too big, results are sporadic: some machines seem to function nicely, while others of similar or identical configuration become hopelessly bogged down. You will need to experiment to find the right size for your listen queue. Version 1.1 of the Netscape Communications and Commerce Servers will never request a listen queue larger than 128.

In kernels that have BSD-based TCP stacks, the size of the listen queue is controlled by the SOMAXCONN parameter. Historically, this has been a #define in the kernel, so if you don't have access to your OS source code, you will probably need to get a vendor patch which will allow you to tune it. In Solaris, this parameter is called tcp_conn_req_max and can be read and written using *ndd*(1M) on /dev/tcp. Sun has chosen to limit the size to which one can raise tcp_conn_req_max using ndd to 32. Contact Sun to find out how to raise this limit further.

Additionally, the kernel on a server machine needs to have enough memory to buffer all the data that it is sending out. In variants of UNIX that use a BSD-based TCP stack, these buffers are called mbufs. The default number of mbufs in most kernels is way too small for TCP traffic of this nature; reconfiguration is usually required. We have found that trial and error is required to find the right number: if 'netstat -m' shows that requests for memory are being denied, you probably need more mbufs. Under IRIX, the parameter you will need to raise is called nm_clusters and it lives in /var/sysgen/master.d/bsd.

TCP employs a mechanism called keepalive that is designed to make sure that when one host of a TCP connection loses contact with its peer host, and either host is waiting for data from its peer, the waiting system does not wait indefinitely for data to arrive. Under the sockets interface, if the socket the system is waiting for is configured to have the SO_KEEPALIVE option turned on, the system will send a keepalive packet to the remote system after it has been waiting for a certain period of time. It will continue sending a packet periodically, and will give up and close the connection if the system does not respond after a certain number of tries.

Many systems provide a mechanism for changing the interval between TCP keepalive probes. Typically, the period of time before a system will send a keepalive packet is measured in hours. This is to make sure that the system does not send large numbers of keepalive packets to hosts which, for example, have idle telnet sessions that simply don't have data to send for long periods of time.

With a web server, an hour is an awfully long time. If a browser does not send information the server is waiting for within a few minutes, it is likely that the remote machine has become unreachable. In the past, router failures were the typical cause of hosts becoming unreachable. In today's Internet, that problem still exists, while at the same time an increasingly large number of users are using a modem with SLIP or PPP as their connection to the Internet. Our experience has shown that these types of connections are unstable, and cause the most situations where a host suddenly becomes silent and unreachable. Most HTTP servers have a timeout built in so that if they have waited for data from a client for a few minutes, they will forcibly close that connection. The situations where a server is not actively waiting for data are the ones most important for keepalive.

If "netstat -an" shows many idle sockets in the kernel or idle HTTP servers waiting for them, you should first check whether your server software sets the SO_KEEPALIVE socket option. The Netscape Communications and Commerce servers do so. The second thing you should check is whether your system allows you to change the interval between keepalive probes. Many systems such as IRIX and Solaris provide mechanisms for changing the keepalive interval to minutes instead of hours. Most systems we've encountered have a default of two hours; we typically truncate it to 15 minutes. If your system is not a dedicated web server system, you should consider keeping the value relatively high so idle telnet sessions don't cause unnecessary network traffic. The third thing you should check with your vendor is whether their TCP implementation allows sockets to time out during the final stages of a TCP close. Certain versions of the BSD TCP code on which many of today's systems are based do not use keepalive timeouts during close situations. This means that a connection to a system that becomes unreachable before it has fully acknowledged the close can stay in your machine's kernel indefinitely. If you see this situation, contact your vendor for a patch.

Your Vendor Is Your Friend; Or, The Value of the Patch

In almost every case, we have gotten quite a bit of value from working directly with the vendor. Since very high volume TCP service of the nature we describe is a fairly new phenomenon, OS vendors are only beginning to adapt their kernels for this.

There are patches to allow the system administrator to increase the listen queue size for IRIX 5.2 and 5.3, as well as enabling the TCP keepalive timer while a socket is in closing states. These patches also fix other problems including a few related to multiprocessor correctness and performance. Contact SGI for the current patch numbers; if you are using a WebFORCE system you should already have

them. With these patches, most parameters the administrator will need to edit are in /var /sysgen/master.d/bsd.

If you are using Solaris 2.3, you will definitely want to get the most recent release of the kernel jumbo patch, number 101318. In general, we've found Solaris 2.4 able to handle much more traffic than even a 2.3 system with scalability patches installed. If upgrading to 2.4 is an option, we highly recommend it especially when your traffic starts to reach the range of multiple hundreds of thousands of hits per day. The 2.4 jumbo patch number 101945 is also recommended, both for security as well as stability reasons.

Logging

Generally, the less information that you need to log, the better performance you will get. We found that by turning off logging entirely, we typically realized a performance gain of about 20%. In the future, many servers will offer alternative log file formats to the current "common log format," which will provide better performance as well as record only the information most important to the site administrator.

Many servers offer the ability to perform reverse DNS lookups on the IP addresses of the clients that access your server. While it is very useful information, having your server do it at run-time tends to be a performance problem. Since many DNS lookups either timeout or are extremely slow, the server then generates extra traffic on the local network, and devotes some (often non-trivial) amount of networking resources to waiting for DNS response packets.

For high-volume logging, syslogd also causes performance problems; we suggest avoiding it. If one is logging 10 connections per second, and each connection causes two pieces of data to be logged (as it does for us), this could mean up to 20 context-switches into syslogd and 20 out of it per second. This overhead is in addition to any logging-related I/O and all processing related to actual content service.

On our site, the logs are rotated once every 24 hours and compressed into a staging directory. A separate UNIX machine inside of our firewall uses an SSLified *rsh*(1) to bring the individual logs to a 4 gigabyte partition where the logs are uncompressed, concatenated and piped to our analysis software. Reverse DNS lookups are done at this point rather than at run time on the server, which allows us to do only one reverse lookup per IP address that connected during that day. Processing and lookups on the logs from all of the machines on our site takes approximately an hour to complete.

A single compressed log file is approximately 70 megabytes, and consequently, we end up with over 250 MB of log files daily. Our method of log manipulation allows for an automated system of backing up and processing a months worth of data with little or no human intervention. Tape backups are generated onto 8mm tape once monthly.

Overall analysis of the log files shows consistent data supporting the following:

a) Peak loads occur between 12 and 3 o'clock PM, PST. Peak connection rates were between 120-140 connections per second, per machine. A second peak of roughly half the amplitude occurs between 5 and 6 o'clock PM, PST.
b) Wednesday is the highest load day of the week, generating more than both weekend days combined.

Equipment

Networks

We found that one UNIX machine doing high-volume content service was about all that an Ethernet could handle. Putting two such machines on a single Ethernet caused the performance of both machines to degrade badly as the net became saturated with traffic and collisions. More analysis of our network data is still needed. We are finding it to be more cost effective to have one Ethernet per host than to purchase FDDI equipment for all of them.

As an aside, we have found SGI's addition of the "-C" switch to *netstat*(1) in IRIX to be extremely useful. It displays the data collected by netstat in a full-screen format which is updated dynamically.

Memory

This is fairly simple: get lots of it. You want to have enough memory both for buffering network data and for your filesystem cache to keep most of the frequently accessed files that it serves in memory. The filesystem read cache hit-rate percentage on our web servers is almost always 90% or above. Most modern UNIXes automatically pick a reasonable size for the buffer cache, but some may require manual tuning. Many OS vendors include useful tools for monitoring your cache hit rates: System V derivatives have sar; we also found HP/UX's *monitor*(1M) and IRIX's *osview*(1) helpful.

Our Typical Configuration

A typical webserver at our site is a workstation class machine (e.g., Sun SPARC 20, SGI Indy, or Pentium P90) running between 128 and 150 processes. For UNIX machines at least, we have found 128 megabytes of memory to be about the most that our machines can use. With this much memory, we have all the network buffer space we need, we get a high filesystem read-cache hit rate, and usually have a few (or even tens of) megabytes to spare (depending on the UNIX version).

Generally one gigabyte or more of disk is necessary. These days, each of our servers generates over 200 megs of log data per day (before compression), and the amount of HTML content we are housing continues to grow. Data from sar and kernel profiling code suggest that our boxes are spending between 5% and 20% of their time waiting for disk I/O. Given the high read-cache hit-rate, we expect that by moving our log files onto a separate fast/wide SCSI drive and experimenting with filesystem parameters, this percentage will decrease fairly significantly.

Miscellaneous Points

In this section, we will discuss a few random things that we have learned during our tenure managing web servers.

A Bit About Security

In addition to the normal security concerns of sites on the Internet[12], web servers have some unique security "opportunities". One of the most notable is CGI programs[13]. These allow the author to add all sorts of interesting functionality to a web site. Unfortunately, they can also be a real security problem: since they generally take data entered by a web user as their input, they need to be very careful about what such data is used for.

If a CGI script takes an email address and hands it off to sendmail on the command line, the script needs to go through and make sure that no unescaped shell characters are given to the shell that might cause it to do something unexpected. Such unexpected interplay between different programs is a common cause of security violations. Since many users who want to provide programmatic functionality on their web pages are not intimately familiar with the ins and outs of UNIX security, one approach to this problem is to simply forbid CGI programs in users' personal web pages.

However, we suggest an alternative: mandate use of taintperl[14] for CGI programs written by users. Perl is already one of the predominant scripting languages used to write CGI programs; it is extremely powerful for manipulating data of all sorts and producing HTML output. taintperl is a version of perl which keeps track of the source of the data that it uses. Any data input by the user is considered by the taintperl interpreter to be tainted, and can't be used for dangerous operations unless explicitly untainted. This means that such scripts can be reasonably easily audited by a security officer by grepping for the untaint command and carefully analyzing the variables on which it is used.

Web Server Heterogeneity

An interesting feature of our site is that it is an ideal testbed for ports of our server to new platforms. Since it was clear to us from the beginning that we would be using it this way, we needed to think about how we would deal with CGI programs and their environment. After some thought, we decided that it just wasn't practical to try to port and test all of our CGI content (and force our users to do the same for their home pages) to every new platform that we wanted to test. It turned out that we were very fortunate to have considered this early, as we eventually ended up testing our Windows NT port on our web site.

We designated a single alias to a machine that would run all of our CGI programs. We then created the guideline that all HTML pages should simply point all CGI script references to this alias. A nice side effect is that this type of content is partitioned to its own machine which can be specifically tailored to CGI service. If the machine pool contains many incompatible machines, this setup avoids having to maintain binaries for CGI programs compiled for each machine. An average day at our site shows 50,000 (out of 7.5 million) requests for CGI scripts (note that this ratio is almost certainly very dependent on the type of content served).

An additional experiment would be to partition other types of content (e.g., graphics) to their own machines in the same way. If necessary, DNS-based load-balancing could be used to spread out load across multiple machines of the same type (e.g., gif.netscape.com could be used to refer to multiple machines whose only purpose in life is to serve GIF files).

FTP vs. HTTP

Both FTP and HTTP offer a easy way to handle file transfer, each with relative strengths.

FTP provides a "busy signal", that is, feedback to the user indicating that a site is currently processing too many transactions. That limit is easily set by the site administrator. HTTP provides a mechanism for this as well, however it is not implemented by many HTTP servers primarily due to the fact that it can be very confusing for users. When a user connects to an FTP site, they are allowed to transfer every document they need in that session. Due to HTTP's stateless nature and the fact that it uses a new connection for each file transfer, a user can easily get an HTML document and then be refused service when asking for the document's inlined images. This makes for a very confusing user experience. It is hoped that future work in HTTP development will help to alleviate this problem. Further work in URL or URN arenas will hopefully provide more formal mechanisms for defining alternative distribution machines.

In a system planning sense, FTP should be considered to be a separate service, and therefore can be cleanly served from a completely different computer. This offers easier log analysis of file delivery vs. html served, and also aids in security efforts. A system running only one service, correctly configured,

is less likely to be breached, and if breached, does not mean loss of security for our entire site.

Performance gains are also likely. Content typically served by FTP is composed of large files, compared to HTTP-served data which is typically designed to be small and quickly accessible by modem users. A document and its inlined images are short enough to be delivered in short periods. Mixing the two different types can make it hard to pin down system bottlenecks, especially if FTP and HTTP are being served from a single machine. Many times the two services will compete for the same resources, making it hard to track down problems in both areas.

While running both FTP and HTTP servers on one machine, we found that 128 HTTP daemon processes and an imposed limit of 50 simultaneous FTP connections was about all a workstation-class system would tolerate. Further growth beyond that would cause each service to be periodically denied network resources. Once separated, however, 250 simultaneous FTP connections on a workstation-class machine was handled easily.

HTTP service, on the other hand, offers a method to gather useful information from the requestor via forms before allowing file transfer to take place. This became a necessity at Netscape, as the encryption technology in our software required a certain amount of legal documentation to be agreed to prior to download.

Conclusion and Future Directions

Although sites such as ours are currently the exception, we expect that they will soon become the rule as the Internet continues its exceedingly rapid growth. Additionally, we expect content to become vastly more dynamic in the future, both on the front end (using mechanisms such as server push[14] and Java[15]) and the backend (where using SQL databases and search engines will become even more common). This promises to provide many new challenges, especially in the area of performance measurement and management.

We hope that the techniques and information in this paper will prove helpful to folks who wish to administer sites providing a very high volume of service.

Acknowledgements

Special thanks to Brendan Eich for his toolsmithing and his willingness to get his hands dirty with the kernel, to Jeff Weinstein for SSLrsh and SSLrdist, to Rod Beckwith for general network magic, and to Gene Tran for help with performance measurement and the SGI kernel profiler. Thanks to Bill Earl, Mukesh Kacker, Mike Karels and Vernon Schryver for TCP bug-fixes and general advice. We also appreciate the input from the folks who took the time to read drafts of our paper.

About the Authors

The authors have been responsible for the design, implementation, and babysitting of the Netscape web site since its inception.

Dan Mosedale <dmose@netscape.com> is Lead UNIX Administrator at Netscape. He has been managing UNIX boxes for long enough to dislike them thoroughly. Dan likes playing around with weird Internet stuff and wrote a FAQ list about getting connected to the MBONE.

William Foss <bill@netscape.com> is Webmaster at Netscape. His current focus is on how to make the site scale even further in an economical fashion. Prior to Netscape, he had the enviable job of playing with large scale UNIX systems and working for Jim Clark at Silicon Graphics.

Rob McCool <robm@netscape.com> is a member of technical staff at Netscape. He designed and implemented the Netsite Communications and Commerce servers. Prior to Netscape he designed, implemented, tested, documented and supported NCSA httpd from its inception through version 1.3.

Bibliography

[1] http://home.netscape.com/comprod/mirror/index.html

[2] http://home.netscape.com

[3] Kwan, T., McGrath, R., & Reed, D., "User Access Patterns to NCSA's World Wide Web Server," http://www-pablo.cs.uiuc.edu/Papers/WWW.ps.Z.

[4] Brisco, T., "DNS Support for Load Balancing," RFC 1794, USC/Information Sciences Institute, April 1995. ftp://ds.internic.net/rfc/rfc1794.txt

[5] Schemers, Roland J. "lbnamed: A Load Balancing Name Server in Perl" LISA IX Conference Proceedings, http://www-leland.stanford.edu/~schemers/docs/lbnamed/lbnamed.html

[6] ftp://prep.ai.mit.edu/pub/gnu/cvs-1.5.tar.gz

[7] Cooper, M., "Overhauling Rdist for the 90's," LISA VI Conference Proceedings, pp. 175-188. ftp://usc.edu/pub/rdist

[8] http://home.netscape.com/info/security-doc.html

[9] Padmanabhan, V., & Mogul, J., "Improving HTTP Latency," Proceedings of the Second International WWW Conference (October 1994), pp. 995-1005. http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/mogul/HTTPLatency.html

[10] Spero, S., "Analysis of HTTP Performance Problems," 1994. http://sunsite.unc.edu/mdma-release/http-prob.html

[11] Viles, C., & French, J., "Availability and Latency of World Wide Web Information servers," Computing Systems, Vol. 8, No 1, pp. 61-91, USENIX Association.
[12] When looking for UNIX security information, a couple of excellent places to start are http://www.alw.nih.gov/Security/security.html and http://www.yahoo.com/Computers_and_Internet/Security_and_Encryption/
[13] http://hoohoo.ncsa.uiuc.edu/cgi/
[14] http://home.netscape.com/assist/net_sites/pushpull.html
[15] http://java.sun.com

Bringing the MBONE Home: Experiences with Internal Use of Multicast-Based Conferencing Tools

Archibald C.R. Mott – cisco Systems, Inc.

ABSTRACT

Some authorities feel that increasing amounts of traffic, rising popularity of multicast-based conferencing tools, and increasing availability of multicast routing technology are signals of impending doom for the Internet's Multicast Backbone (MBONE). However, they are also clear signals that there is a very real demand for the services of which they are a result: real-time, multipoint, multimedia interaction between network users. Whether or not the utility of the MBONE itself is drawing to a close, it is a simple step to draw parallels between the demands of users on the Internet and the demands of users on a large corporate network, and to see that the same tools in use on the MBONE can be used to provide an important service in a corporate network environment.

This paper will describe the implementation of a widely distributed conferencing system based on IP multicast networking and freely available conferencing tools. It will describe the network topology and routing technology employed, the scope of the system, some challenges encountered in implementing the system, the tools used in the implementation, real examples of the use of the system, future plans for the system and an exploration of some potential pitfalls of the system. The information presented in this paper is based on experiences gained in deploying the system on the Engineering departmental networks at Cisco Systems. Opinions expressed in this paper are those of the author, and do not necessarily reflect the opinions of Cisco Systems.

IP Multicast Networking

The features of IP multicast [1] which make it indispensable as a transport for conferencing applications traffic on the Internet make it equally indispensable on a corporate network: use of multicast enables wide distribution of the traffic over backbone networks with a minimum of replication; large numbers of hosts on a single network are able to simultaneously receive a single stream of multicast traffic; judicious configuration and use of multicast routers enables networks with no participants to not receive the traffic at all.

What Is IP Multicast?

IP multicast networking uses a method of addressing IP packets so that their destination is a "group" of hosts rather than a single host or a broadcast. A group can contain zero or more hosts, and hosts can dynamically join or leave a group at any time. These groups are addressed using class D IP addresses, which have the four high-order bits set to "1110". In dotted decimal notation this means the range of group addresses is from 224.0.0.0 to 239.255.255.255. Using these group addresses it is possible to deliver data to multiple hosts on a network with a single stream of data for the entire group, rather than duplicating the data for each host that "wants" to receive it. 256 addresses in the low-range of the class D address space are reserved. For instance, 224.0.0.1 is the group address for "all" IP hosts (which in effect is all multicast capable hosts on a single network); 224.0.0.2 is the group address for all multicast routers.

Using IP multicast is similar to using IP broadcast in that it is a method of contacting multiple hosts simultaneously, and therefore provides an efficient method of delivering data to many hosts. Unlike broadcast traffic on a network, hosts are not required to "listen" to multicast traffic since the joining of a group is voluntary. Also unlike broadcast traffic (in most cases) multicast traffic can easily be routed between multiple networks.

For the purposes of implementing the ability to participate in multicast-based conferencing, a host should have full (level 2) support of IP multicast, which includes the ability to send and receive multicast traffic, and the ability to join and leave host groups. These abilities are accomplished using the Internet Group Management Protocol (IGMP) and extensions to a host's network interface code. The networking extensions are required to allow a host to receive data addressed to any host-groups to which it is joined, rather than data addressed only to that host or to a broadcast address. The IGMP extensions allow the host to inform any multicast routers on its network of the host's membership in any host groups.

Routing IP Multicast Traffic

The most commonly used methods for routing IP multicast traffic are Distance Vector Multicast Routing Protocol [2] (DVMRP) and Protocol Independent Multicast [3] (PIM) routing. Another method available is Multicast OSPF [4] (MOSPF). Of these routing methods DVMRP is probably the most widely used since it has been available for the longest time, and can be run using the multicast routing daemon (mrouted) on a large number of UNIX platforms. Various router vendors have implemented one or more of these multicast routing methods. Most multicast routing methods include mechanisms for "tunneling" multi-cast traffic through non-multicast-capable routers to allow seemingly continuous multicast connectivity across segments of networks where multicast traffic is not supported. Tunneling is usually accomplished by encapsulating multicast traffic inside of regular IP unicast packets to allow it to be routed.

Multicast on our Engineering Networks

Cisco Engineering's use of multicast networking in our production networks was driven by a need to test our multicast routing code. One of the easiest ways to perform this testing was to use existing applications that generated large amounts of multicast data. Since the MBONE was already being used to multicast audio, video, whiteboard and other data, and since tools for sending and receiving this sort of traffic had been developed and used on the MBONE, it was easy to decide that building an internal equivalent of the MBONE would be a good beginning for our testing. We started to install multicast capability on a subset of our desktop workstations, and receiving broadcasts from the MBONE. The number of networks which included multicast support started out quite small, but as more people became aware of the existence of the internal availability of the MBONE, the demand to deploy multicast routing capability and conferencing tools widely became overwhelmingly apparent.

Deploying Multicast Routing

Like most MBONE users our original routing configuration was based primarily on DVMRP routers using the 'mrouted' program, and connected to each other via DVMRP tunnels. The DVMRP routers used were Sun machines. We also used a DVMRP tunnel to our Internet Service Provider as our connection to the MBONE, as well as DVMRP tunnels interconnecting the networks where we wanted to deliver multicast traffic.

As development of Cisco's implementation of PIM routing continued, we started using PIM routers to support various networks. We replaced many of the DVMRP routers which were supporting test networks, and added test networks using PIM support. We gradually deployed PIM routers on a subset of production nets, but maintained DVMRP connection to Internet. As our comfort level with multicast routing grew, we deployed it on more of our production networks until it became a 'default' feature on Engineering networks in our environment. When our Internet Service Provider was ready, we finally moved to all PIM routing, including our Internet connection

Current Topology

The topology now carrying multicast traffic at our site is fairly complex. Conferencing traffic is carried on Ethernet, FDDI and ATM backbone networks between buildings, on switched and unswitched 10BaseT networks and CDDI networks to desktop workstations, and on a variety of WAN media including ISDN and frame-relay to remote offices and telecommuter sites; routing is handled through the PIM implementation on Cisco routers.

Conferencing Tools

One of the challenges facing the Systems Administrator who would like to provide on-line conferencing services to the user community is that of doing so across heterogeneous platforms. A variety of commercial conferencing packages is available, but few of them offer the heterogeneous support desired. Fortunately a suite highly usable conferencing tools has been made available for a wide range of UNIX platforms. Another challenge the Systems Administrator encounters is that of providing the same services available to UNIX users to the non-UNIX user community. Another publicly available package has given us the ability to allow Macintosh and IBM PC users to participate at least partially in our conferencing system.

Tools from the MBONE

The first applications we chose to use for conferencing were the suite of tools that have been traditionally used by MBONE participants: vat, the Visual Audio Tool; nv, the Network Video Tool; wb, a shared whiteboard tool; and sd, the Session Director tool, which is used to send and receive "advertisements" for multicast groups and to control the applications needed to participate in them. These tools operate under the X Window System.

The Visual Audio Tool, vat, was developed at Lawrence Berkeley Laboratories. It allows users to participate in many-to-many audio conferences using multicast. It can also be employed in a point-to-point mode without multicast. Vat allows a number of configuration options to be set, including the selection of audio input and output devices, selection of various audio encoding schemes, manipulation of input and output levels, and the use of a DES encryption key to provide "some measure of privacy".

The Network Video Tool, nv, was developed primarily at Xerox PARC. It also allows many-to-

many and point-to-point conferencing. Nv provides the ability to capture video from a limited number of frame grabbers as well as the ability to use a "screen grabber" to send images from a particular window or region on the sender's screen. No special hardware is required for nv to receive video. Controls available with nv include a transmission bandwidth limiter, settings for the encoding of the video, image size and color controls for both the image to be transmitted and the images being received, and grabber controls to allow selection of an input device. At this time, no encryption option is available.

The whiteboard tool, wb, was also developed at Lawrence Berkeley Laboratories. It provides conferencing users with a shared "drawing" surface that allows freehand drawing as well as the entering of text. It is also possible to import ASCII text files. If used in conjunction with Display Postscript capable X servers or with Ghostscript, wb is also capable of displaying imported postscript images. An additional tool, wbimport, allows wb users to configure whiteboard "slide shows". There are many controls available to wb users including "pen" color, whiteboard orientation, line smoothing, participant muting to remove extraneous input from the whiteboard surface and others. Like vat, wb supports DES encryption.

The Session Directory, sd, is another tool that was developed at Lawrence Berkeley Labs. Sd uses multicast to send and receive host-group 'advertisements', as well as having the capability to control the various conferencing applications already discussed, and some tools not discussed in this paper. Sd acts as the "glue" which takes various individual conferencing tools and turns them into a coherent conferencing software system. When started, sd listens to the host group address 224.2.127.255 for session advertisements. These advertisements contain information about host groups such as the address of the group, a name and description of the host group, what media are being carried by the host group (audio, video, whiteboard, etc.), port numbers associated with the various media, and a time-to-live value. As sd hears group advertisements, it caches the information. This caching allows sd to provide useful, though possibly outdated information to the user at its next invocation. In addition to listening to advertisements, sd can also be used to create advertisements for groups. Sd is a Tcl-based application, and uses a configuration file which controls its behavior when launching applications. This configuration allows the user to preset options for the conferencing tools, such as how the user's name will be displayed in participant information displays. It also makes sd usable with virtually any conferencing tool because the user can choose which audio, video, and whiteboard applications they want to use. It is up to the user, however, to make sure that their application of choice will be interoperable with other users' software.

Evaluation of other tools

When it was decided that the Engineering Computer Services group would provide desktop conferencing as a supported service to the engineering community at Cisco, we started to look beyond the existing freely available tools to see if there were packages that better met our needs. The number of commercial, desktop system-based conferencing packages is growing. As can be expected, many of these commercial offerings are more polished and have apparently fuller feature sets than the free software, but all of the packages we reviewed lacked some essential features. The most common problem we encountered was that many of the packages were platform specific. Another common problem was that some of the applications did not use multicast.

We eventually decided to continue with our use of the above mentioned freely available tools, but added the Multi-Media Conference Controller (MMCC) to the list of tools we offered to our users. MMCC is analogous to sd in that it acts as a controller of audio, video and whiteboard applications. However where sd uses an advertisement scheme to inform users of the existence of host groups which may be joined, MMCC allows a user to build a list of conference 'invitees' and then notifies the invitees of the existence of the host group. MMCC also allows users to maintain a "phone book" of users who are frequently involved in conferences.

There were some functionality trade-offs involved in the choice to use freely available software. For instance some of the commercial whiteboard applications are much more flexible than wb. Some of the commercial conferencing packages allow the sharing of applications programs so that users at multiple workstations can collaboratively use one copy of a non-conferencing tool such as a CAD application.

Support of Non-UNIX platforms

Another one of the challenges we had to address in our implementation of conferencing systems throughout our engineering department is that of providing conferencing services to our users who are not using UNIX based systems. Once again, a number of possible solutions presented themselves, but the need for interoperability with the UNIX based solutions led us to the use of Cornell University's CU-SeeMe software.

Multiuser conferencing with CU-SeeMe is accomplished by the use of the CU-SeeMe tool at the desktop in concert with an application known as a "reflector". The reflector process allows users of CU-SeeMe, which is essentially a point-to-point conferencing tool, to have the reflector as an endpoint. The reflector then handles to distribution of incoming traffic to the CU-SeeMe users connected to it. An important feature of the reflector process is that it can be configured to join a host-group and then

gateway audio and video traffic to many CU-SeeMe clients. These clients unfortunately do not currently have the capability to use IP multicast which means that there is a geometrically increasing amount of traffic with the increasing number of CU-SeeMe users on a network. Currently, we have located a reflector centrally in our network topology, so the traffic for CU-SeeMe traverses a small number of end-user networks and then goes onto our comparatively high-bandwidth backbone, where the traffic's impact is minimal. Since the reflectors themselves are capable of joining multicast host-groups, it would be wiser to deploy a larger number of reflectors closer to the end users; this will most likely be part of our near future implementation.

Applications of Conferencing Tools

We have had success using these conferencing tools in a number of ways. The principal benefit derived from the use of such tools is that of bringing together geographically separated groups or individuals who might normally interact via electronic mail or telephone calls, and allowing them to share more than just one form of data. We are continuing to come up with new methods of applying conferencing technology.

Weekly Nerd Lunch Lecture Series

Cisco has a long standing tradition of conducting weekly lunchtime lectures which we call "Nerd Lunches" in which one of our engineers or a visiting lecturer will discuss technical issues. As the company has continued to grow both in number of employees and in geographical scope it has become increasingly difficult to accommodate everyone who wants to attend these meetings. On-line conferencing has provided a method for us to allow anyone who would like to attend to do so from their desktop whether it is across the hall or across the continent from the lecture room. One trick we have learned to minimize interruptions is to mute the speaker on the workstation which is acting as the 'transmitter' for the lecture and to have remote attendees type their questions on a shared whiteboard.

Broadcast Offsite Events Over ISDN

Occasionally Cisco uses offsite facilities to hold events which we want to broadcast using on-line conferencing. We have accomplished this by using a portable UNIX machine in conjunction with a router equipped with multiple Basic Rate ISDN (BRI) interfaces. By using the multiple BRI interfaces, we can use two ISDN lines with two 64 kbps B-Channels each for a total (theoretical) throughput of 256 kbps. While this is not enough band-width to provide the same sort of audio and video quality as a direct Ethernet (or better) connection, we have still been able to transmit audio using dvi4 encoding at approximately 32 kbps and video using the remaining bandwidth at a frame rate of up to 5 frames per second. This provides a setup which can be easily arranged at almost any off-site location in our area through the simple installation of a pair of ISDN lines.

Training

The use of MBONE conferencing tools for training has allowed us to deliver training to multiple sites without sending trainers to all of those sites. Recently we have held training classes at our headquarters site on San Jose, and by broadcasting over our network allowed Customer Engineers at our Research Triangle Park site to attend. Generally, a video crew is hired to record these training sessions. The video crew brings their own equipment including video mixers and multiple cameras. By taking our video feed from the video crew's mixer, we were able to transmit video using multiple camera angles, picture-in-picture video and other "professional" video effects.

Sysadmin "Intercom" Application

One of the host-groups we carry via multicast is the "Engineering Computer Services Intercom", which is an audio-only group. The idea behind this group is that the systems administration team joins the group at their workstations, and it can be used to ask and answer questions. This is especially helpful since the group is not entirely co-located. Another potential advantage would be for a systems administrator at a user's workstation to join the group if they were having trouble solving the user's problem so they can get assistance. This application has not been a complete success. One reason for this may be that, since we would prefer that the user community at large not join this particular group, we do not advertise it through sd. Manually joining audio groups with vat is not necessarily difficult, but can be cumbersome. Many of the sysadmins have handled this by aliasing the vat command.

MMCC as a Multimedia Phone System

As mentioned before, the Multi-Media Conference Controller program provides a method of creating host-groups without advertising them using sd. Instead, the group-originator's copy of mmcc will attempt to contact a copy of mmcc on each user's workstation that has been invited to join the group. In an audio-only mode, this can act as a "conference-call" system, or be used to make point-to-point "phone" calls. However mmcc has the added advantage of being able to use video and whiteboard as well. The drawback however is that all invitees to a group must also be running mmcc for its use to be successful.

Challenges

Dealing with New Technology

As with the application of any new technology there was a learning period while we found how best to deploy multicast support to our workstation users,

multicast routing to our networks, and conferencing applications to our users. We have spent time with the developers of Cisco's version of PIM trying to track down configuration errors, and trying to come up with designs that will both provide reliable delivery of multicast to our users' networks and provide useful testing and debugging data to the developers. At times, these goals seem to be mutually exclusive.

We have also offered training sessions in the use and application of multicast conferencing tools to help familiarize our users with our conferencing offerings. There has been a minor increase in our user support requests that can be correlated with the introduction of these tools. There has also been a small additional burden to our support load that is related to policing the use of our multicast capability. For instance, users have occasionally forgotten to mute their microphones and have ended up inadvertently sending unwanted audio to the network.

Fear of impact

Another issue we have had to address is the fear that our production networks would become bogged down from carrying too much multicast traffic as more users started joining various multicast groups. While we have experienced periods of large amounts of multicast traffic on some of our nets, for the most part these have been minimized by training our user community to be judicious with the bandwidth they consume for conferencing purposes. Another strategy we have used is to advertise audio, video and whiteboard sessions as separate host groups, allowing users to choose which data they will receive. This has proven especially helpful to our telecommuting users who connect via ISDN or 56kbps Frame Relay lines.

High User Expectations

Possibly the most significant challenge we have encountered is that of high user expectations. Many users expect that desktop videoconferencing should include high quality images at a high frame rate, 'just like television'. In our training sessions, we try to make clear to the users that most of the content of many conference groups can be contained in the audio and whiteboard media, and that the video is "icing on the cake". There are however times when 30 frame per second video would be enjoyable. There are factors that make this difficult to deliver. First, even on a switched Ethernet network it can be difficult to deliver video at those kinds of speeds, especially without highly efficient video compression and decompression facilities; Second, at this time we have not widely deployed hardware that is capable of capturing video at high frame rates.

Reflectors Only Handle One Group

Currently, the CU-SeeMe reflector process is capable of reflecting only one set of audio and video from multicast sources to CU-SeeMe users. While the audio and video sources may be carried on different host-group addresses (e.g. if audio and video are being delivered on different multicast addresses), this means that where the UNIX user is able to simultaneously join an arbitrary number of "conferences" at will from a list of advertisements, the CU-SeeMe user is capable only of joining the conference being carried by the reflector to which they connect.

Scope of the System

Systems Supported

We currently offer conferencing support on Sun, HP and SGI workstations using the MBONE tools, and on Macintoshes using CU-SeeMe. A beta version of CU-SeeMe is being evaluated by our (growing) population of PC users.

Network Scope

Currently, Cisco's Engineering Computer Services department is deploying multicast routing to support all of our production engineering LAN's. At this point, more than 50 LAN segments are supported using PIM routing. We also are routing multicast traffic over WAN links (T1 or faster) to support remote sites. We are also implementing multicast routing support on ISDN and Frame-Relay networks to support telecommuters.

Geographic Scope

The geographic scope of the multicast network now covers 8 buildings at our headquarters location; Our site in Research Triangle Park, North Carolina; Our ATM Business Unit site in Billerica, Massachusetts; and many telecommuters' homes in locations around the United States. Many hundreds of people are currently have access to the multicast networks, and conferencing applications which use them.

Current Shortcomings and Challenges

Security

Access to the MBONE is in general fairly risk-free, however as more of our users get access to the resource, we have endeavored to minimize the possibility of sensitive information contained in conferences being allowed past our firewalls. We have employed a multipart strategy to accomplish this: assigning low Time-To-Live (TTL) values to our host groups prevents them from traversing large numbers of multicast routers; The use of thresholds on our firewalls prevents any traffic with TTL below a specified value from being forwarded; and filters on the routers to prevent specific host-groups from traversing the firewalls. In conjunction with user training and occasional monitoring of internal host groups, the enforcing of the use of low TTL's has not proven to be difficult. We are also filtering access to our internal CU-SeeMe reflector to prevent external sources from accessing it. These security

measures still allow our users to participate in MBONE conferences, while providing a reasonably secure internal conferencing environment.

There are discussions in the IETF to implement "scoped addressing" for multicast host groups. This feature would allow certain ranges of addresses to be confined within predefined site boundaries. This should provide a more convenient method of containing multicast traffic, for the purposes of both security and restricting bandwidth consumption on the MBONE.

Etiquette Implications

Trashing the MBONE

Many sites would like to be able to use the MBONE as a carrier for communication with customers and clients. Many routers in the MBONE currently enforce rate limiting that makes the effective capacity of the MBONE about 500 kbps. Given that the average MBONE conference consumes roughly 40 to 150 kbps it is apparent that there is not much bandwidth available for "private" use in an environment that is governed by the principal of providing "the greatest good for the greatest number", and is managed by rough consensus. As the bandwidth available for multicast traffic increases and the multicast topology of the MBONE is optimized by the use of native multicast routing (as opposed to routing tunneled multicast traffic) this may become less of an issue. In the meantime, workarounds need to be developed. It is the author's opinion that the use of multiple, loadbalanced ISDN interfaces between sites is one such workaround.

There goes the Neighborhood

One of the annoying consequences of a widely deployed conferencing system is the increased noise level in the work environment. This is compounded by the "echo effect" which can be caused by multiple machines in an area receiving the same multicast data, but as a result of different machine speeds and load levels reproducing audio at different times. Noise level is also increased by conferencing users speaking to their microphones. Anyone who considers deploying conferencing ability should also consider investing in headsets for their conferencing users.

Least Common Denominators

In order for CU-SeeMe users to receive video from nv users, the nv users must originate their video in CU-SeeMe format, which at this time is by definition greyscale video.

We also have many users of X terminals. While some X terminal vendors have been adding multimedia abilities to their products, we currently do not have any that are capable of taking advantage of nv- or CU-SeeMe-based video or vat based audio.

Future Plans

Virtual Conference Rooms

For some time now we have been advertising internal audio, video and whiteboard groups that any of our users can access. It has been suggested that we should create "Virtual Conference Room" groups that can be reserved for specific meetings.

Remote Camera Control

In real conference rooms where on-line conferencing systems have been installed we are installing large displays and video cameras that can be controlled through a serial data port. We will develop a program for controlling the camera's tilt, pan, zoom and focus so that a remote station can choose what is being shown in the video.

More Reflectors

In order to better serve the CU-SeeMe user community, and to more efficiently distribute conferencing traffic, more CU-SeeMe reflectors need to be deployed. This is especially crucial for remote sites connected to the headquarters network via T1 lines.

Resource Reservation

As methods such as the Resource Reservation Protocol (RSVP) become mature, we will employ them to provide guaranteed bandwidth for conferencing.

On-Demand Video/Audio

One feature which our conferencing system does not address and is already in demand by our users is the ability to store video and audio for on-demand retrieval and display. The potential resource costs are high for this service because of the storage and bandwidth requirements. We are currently evaluating on-demand server packages.

More More MORE!

In order to satisfy the demands of the user community, it will be necessary to continually upgrade the capability of our conferencing system. Bandwidth is probably the most important consideration, especially as the number of conferencing groups increases. The number of users, thanks to the advantages of multicast traffic, is less of an issue. We will also need to start using faster frame grabbing hardware to provide better video service. We will also need to continually evaluate new packages for providing conferencing services.

Getting the Software

The wb, vat and sd packages were developed at Lawrence Berkeley Laboratories. They are available for anonymous ftp at ftp.ee.lbl.gov in the /conferencing directory

nv was developed by Ron Frederick at Xerox PARC. It is available fro anonymous ftp from parcftp.xerox.com in the /pub/net-research directory.

IP Multicast support for SunOS machines is available for ftp from parcftp.xerox.com in the /pub/net-research/ipmulti directory.

mrouted is available for ftp from parcftp.xerox.com in the /pub/net-research/ipmulti directory.

CU-SeeMe and the CU-SeeMe reflector were developed at Cornell University. They are available for anonymous ftp from gated.cornell.edu in the /pub/video directory.

References

[1] Deering, S.: RFC1112.

[2] Waitzman, D.; Partridge, C., Deering, S. BBN STC: RFC 1075.

[3] Deering, S.; Estrin, D.; Farinacci, D; Jacobson, V.; Liu, C.; Wei, L.: Protocol Independent Multicast: Protocol Specification.

[4] Moy, J.: RFC 1584.

Author Information

Arch Mott is a Senior Systems Administrator at Cisco Systems, Inc. He can be reached via U.S. Mail at Cisco Systems, 170 West Tasman Drive, San Jose, CA, 95006 or electronically arch@cisco.com

LACHESIS: A Tool for Benchmarking Internet Service Providers

Jeff Sedayao and Kotaro Akita – Intel Corporation

ABSTRACT

Internet access is increasingly critical to organizations and individuals [1]. With the current boom in Internet Service Providers (ISPs), how does one judge one vendor from another? LACHESIS[1] is a tool that provides a way to benchmark ISPs. LACHESIS takes a list of prominent Internet "Landmarks" and determines the packet loss and network latency involved in reaching those landmarks. Throughput was rejected as a factor. Several studies indicate that network latency is a critical factor in World Wide Web (WWW) performance [2-5]. The default set of LACHESIS landmarks (landmarks used are customizable) includes the Domain Name Service (DNS) root servers, well known FTP servers, and popular WWW servers. LACHESIS is implemented as a PERL script wrapped around FPING. The LACHESIS tool encourages ISPs to have good interconnectivity with other ISPs. It also encourages ISPs to have plenty of capacity and not to drop packets. LACHESIS has the potential to swamp landmarks with ICMP packets (used by FPING), but this can be dealt with by filtering out ICMP from abusive hosts. ISPs can cheat by favoring ICMP packets. Future plans include a Winsock implementation so that individual SLIP/ PPP Internet subscribers can run their own benchmarks.

Introduction

Internet access is becoming more and more critical to more and more organizations and individuals. Ignoring events on the Internet can have serious consequences [1]. There is a boom in Internet service providers (ISPs), ranging from local phone companies (e.g., Pacific Bell or Ameritech), long distance companies (e.g., AT&T, MCI, and Sprint) or On-line services companies (e.g., Compuserve, Prodigy, and America On-line), and start-ups (e.g., Internex, PSI, and UUNET). But how does one judge one ISP from another? What metrics does one use? What tools are available for measuring "performance" from one another? LACHESIS is a tool that provides benchmarks for Internet service.

The first part of this paper describes the LACHESIS approach. The next section discusses how LACHESIS is implemented. Actual results are listed after that. This section focuses on the experiences and implications of LACHESIS. The paper concludes with a discussion of future work, and information on how to get LACHESIS.

The Lachesis Approach

LACHESIS's purpose is to measure the performance of an Internet Service Provider. Performance can mean many things. The time to transfer a file is one measure, while the "responsiveness" of an interactive remote login session is another. The time to call up a Web page is still another.

LACHESIS concentrates on two aspects of Internet performance – packet loss and network delay. If an ISP drops many packets, it will clearly take longer to transmit data or do remote operations because packets need to be retransmitted. Network delay is the time it takes for packets to go through a network. Studies show that World Wide Web traffic is particularly sensitive to delay [2-5]. Domain Name Service (DNS), a service critical to Internet applications, can also be negatively impacted by network latency. Many applications simply idle while waiting for DNS information. Adding network delay to DNS query times only makes things worse.

Why not concentrate on throughput? There are a number of reasons that packet loss and network delay are more critical than throughput. First, throughput will have an absolute upper limit determined by the size of the connection. Having a T3 (45 Megabit) Internet connection will yield vastly different results than from having a 14.4 Kilobit SLIP connection. Second, measuring throughput requires that you have a significant amount of data to move to or from some Internet system. It is not always possible to have this. Third, as mentioned above, applications like WWW are very sensitive to delay and do not use all of the available bandwidth.

[1] In Greek Mythology, LACHESIS was one of the Fates, the three goddesses who determine the string of life. KLOTHOS spun the string of life, LACHESIS measured it, and ATROPOS cut it. [12]

In the long run, as larger and larger amounts of bandwidth to the Internet become cheaper and cheaper, the costs of network delays become higher and higher. Consider the cost of 1 second of network delay. A 45 Megabit T3 connection will waste more potential bandwidth waiting for 1 second delay than will a 14.4 Kilobit connection. Protocols that do format and parameter negotiation are particularly vulnerable to network delays.

When measuring packet loss, we need some targets to measure packet loss. LACHESIS uses the concept of LANDMARKS. Landmarks are notable sites on the Internet. The default landmarks for LACHESIS are the root name servers, popular FTP servers, and popular WWW sites. There are landmarks from around the world to get a more complete picture of an ISP's connectivity. LACHESIS users can configure their own landmarks depending on their own usage patterns. This way, Internet users or organizations can pick an Internet vendor optimized to their particular usage patterns.

LACHESIS is implemented as a PERL [6] script wrapped around a modified version of Stanford University's FPING program. Packet loss and packet round trip times are generated from FPING. FPING uses the ICMP echo [7] to measure network latency. It has been pointed out that PING was not designed for measuring network performance and that different routers may handle ICMP packets in different ways [8]. While this is true, no other metric or protocol feature works with generic landmarks picked by a consumer.

How do we intend for LACHESIS to be used? We envision that organizations with current Internet access could run LACHESIS periodically against their favorite LANDMARKS. Data from these runs could be used to identify problem periods and get a feel for general performance through an Internet vendor. We also envision that when an organization is looking to procure Internet access, they could run LACHESIS from either the ISPs' local pops or at one of the ISPs' local customers. They could then evaluate the ISP from the resulting LACHESIS runs.

LACHESIS Implementation Notes

As mentioned above, LACHESIS is a PERL script wrapped around Stanford University's FPING program. LACHESIS is run periodically, and the packet loss, delay, and other statistics are logged, accompanied with a time stamp. To get a graphical representation of the data obtained by LACHESIS, a separate program was written to transform the data into World Wide Web [9] viewable graphs. The program GRAPHLACHESIS takes in the LACHESIS log file, parses the data, and calls upon another program which produces graphs. People can

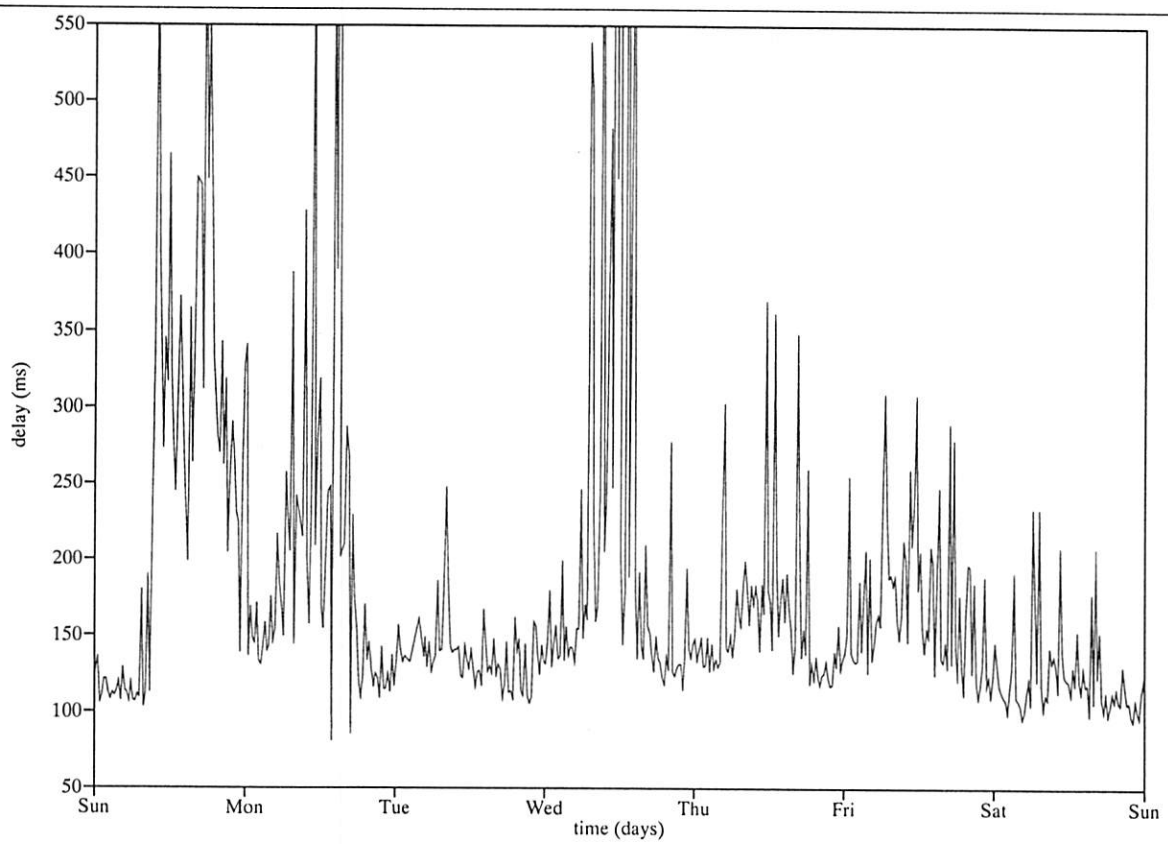


Figure 1: Delay through provider

now access LACHESIS data easily through their favorite Web browser. Nearly real-time analysis and monitoring of ISP performance has been made possible.

Each line of the log file shows when data was collected along with the values for five parameters: network delay, packet loss, number of hosts, hosts unreachable, and hosts unknown. For each of these dependent variables, GRAPHLACHESIS generates a file in the WWW's HTML [9] format. Each of those files are four graphs showing data for the current day, the current week, the previous week, and long term historical trend. Figure 1 shows a graph of delay through an ISP provider during Intel work week 27. Figure 2 is an example of graph of packet loss during that period.

GRAPHLACHESIS parses the log file data and creates individual files for each of the five parameters. The individual files are then fed to a program called WEBGRAPH. WEBGRAPH is a generic graphing package that reads in any single set of data in the form "x, y" and produces a graph that is automatically appended to a specified HTML document. GRAPHLACHESIS calls upon WEBGRAPH repeatedly, each time appending a graph to the appropriate WWW page.

WEBGRAPH was deliberately kept a separate program from GRAPHLACHESIS (as opposed to being a subroutine in GRAPHLACHESIS) because we wanted to have WEBGRAPH available as a stand-alone general purpose graphing package. WEBGRAPH allows the user to control many aspects of the output graph (such as the title, labels, axes ranges, tic marks, and plot style) all from the command line. This way, complete graph generation can be executed in one step. GRAPHLACHESIS takes advantage of this flexibility and demonstrates the usefulness of WEBGRAPH as a generic graphing package.

GRAPHLACHESIS and WEBGRAPH are both written in PERL. The former wraps around the latter, and the latter further wraps around two programs: GNUPLOT [10] and PPMTOGIF from the PBMPLUS package [11]. GNUPLOT plots the data and produces graphs in PBM (Portable Bit Map) format, and PPMTOGIF transforms the PBM graphs into GIF format, making it presentable to the WWW.

Results and Implications

We ran LACHESIS for a number of months against a single Internet vendor. LACHESIS proved useful in recording problems with Internet access. Figure 1 shows a graph of delay during a particularly bad week while figure 2 shows packet loss over this period. Our ISP was having problems with their

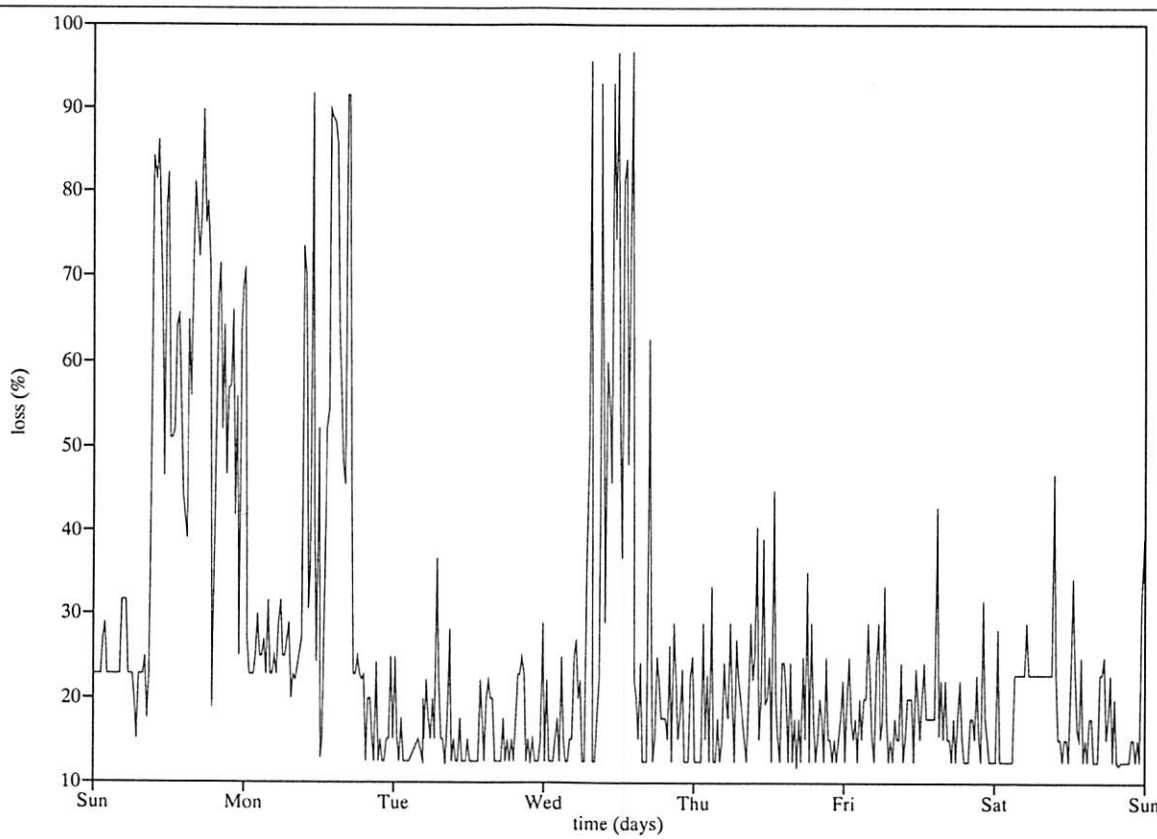


Figure 2: Packet Loss

backbone during this week. Mid-Monday and mid-Wednesday were particularly bad. Note the high delay for systems that could be reached, and the high packet loss (nearly 100%) during those periods. LACHESIS enables us to capture these periods of instability.

Sampling frequency needs to be selected carefully. One common ISP problem we encountered is having our default routes appear and disappear. Since we have multiple connections to the Internet, routes to the Internet and to Intel would be recomputed if our default route disappeared. It would take about 10 minutes for the routes to resolve inside and outside of the Internet. If the route reappeared, it would take another 10 minutes to resolve back again. During these 10 minute intervals, Internet connectivity would be lost. Since we ran LACHESIS every 20 minutes, we lost visibility into these route flaps. In order to catch events such as this, LACHESIS needs to run at least twice as frequently as the length of the event to be monitored.

One interesting suggestion was to run LACHESIS against our own Internet resources (such as the corporate WWW server). This would help us determine our own performance relative on the Internet as well as detect problems in our Internet connections and servers.

What does LACHESIS imply for Internet vendors? To get delays as low as possible, ISPs will need to be well interconnected to other ISPs. Landmarks that are on different ISPs will really bring this out. Vendors who route all of their Inter-ISP traffic through highly congested traffic exchange points will fare poorly using LACHESIS. ISPs must have low internal delay, and routes have to be sensible. ISPs who route traffic between neighboring states across North America and back will not fare well under LACHESIS. ISPs must not drop packets (because of overloaded routers, lines, etc.) because packet loss is measured. ISPs could cheat by letting ICMP packets go through at higher rate, but this is unlikely as it would affect other more important traffic on their networks.

LACHESIS poses a few problems. Will LANDMARKS be flooded by LACHESIS users? We don't think so. Abusers can easily be cut off with router access lists. Another way for LANDMARKS to handle this is to set up "sign posts." These sign posts would be special systems designated to respond to LACHESIS and other applications' pings.

Another problem results from having sites or organizations with multiple Internet connections. If one ISP loses connectivity, LACHESIS starts measuring whatever ISP takes over. The solution to that is to have LACHESIS measuring take place from segments that only route through a particular Internet provider.

One very useful package to fall out of the LACHESIS work is the WEBGRAPH program. We will use this package for plotting all kinds of data and presenting that data on the Web. Using the Web makes LACHESIS information available to a very wide audience. Users running on Intel Architecture PCs to Unix workstations can see the data.

Conclusions and Next Steps

LACHESIS has proven to be a useful tool despite its simplicity. Measuring delay and packet loss is useful for evaluating and benchmarking Internet vendors. Data presented on the Web can reach a wide audience in almost real-time. LACHESIS currently runs on BSDI Unix and SunOS. Future plans include a WINSOCK version for Microsoft Windows (TM) is also being planned to enable individual Internet subscribers to benchmark their own providers. Another idea that we are considering is to apply statistical process control methodologies to LACHESIS data and graphs. We would then benchmark and contact ISPs when they are "out of control".

Acknowledgements

We would like to thank Darci Chapman for ideas about using LACHESIS to benchmark ourselves. Thanks should also go to Cindy Bickerstaff for some ideas on statistics.

Author Information

Jeff Sedayao received a B.S.E. in Computer Science from Princeton University in 1986 and a M.S. in Computer Science from the University of California at Berkeley in 1989. He has worked at Intel Corporation since 1986, spending most of his time running Intel's main internet gateway. Reach him at Intel Corporation; SC9-37; 2250 Mission College Blvd; Santa Clara, CA 95052-8119. Reach him electronically at sedayao@argus.intel.com.

Kotaro Akita is a Junior at Princeton University studying electrical engineering. He plans to concentrate in communication networks and also receive a certificate in Engineering & Management Systems upon graduation. This summer he worked at Intel Corporation for twelve weeks for Jeff Sedayao. Reach him at 1903 Hall #504; Princeton University; Princeton, NJ 08544. Reach him electronically at KAkita@Princeton.EDU.

References

[1] Suzanne Johnson. "Internet Affects the Corporation: Experiences from Eight Years of Connectivity." *Proceedings of INET 95*, Honolulu, June 1995.

[2] Venkata N. Padmanabhan and Jeffrey C. Mogul. "Improving HTTP Latency." *Proceedings of the Second International World Wide*

Web Conference, pages 995-1005, Chicago, October 1994.

[3] Simon Spero. "Analysis of HTTP Performance Problems.´ http://elanor.oit.unc.edu/http-prob.html.

[4] Jeff Sedayao. "World Wide Web Network Traffic Patterns." *Proceedings of Spring COMPCON 95*. San Francisco, March 1995.

[5] Jeff Sedayao. "Mosaic will kill my Network!" *Proceedings of the Second International World Wide Web Conference*, pages 1029-1038, Chicago, October 1994.

[6] Larry Wall and Randal L. Schwartz. *Programming PERL*. O'Reilly & Associates, Inc., Sebastopol, CA, 1991.

[7] J. Postel. "Internet Control Message Protocol – DARPA Internet Program Protcol Specification," *RFC 792*, USC/Information Sciences Institute, September 1981.

[8] Stan Barber. "Minutes from the April 1995 Danvers IETF (IP Provider Metrics BOF)."

[9] Tim Berners-Lee, R. Cailiau, A. Luotonen, H. Nielsen, and A. Secret. "The World Wide Web." *Communications of the ACM*. 37(8):76-82. August 1994.

[10] Thomas Williams and Colin Kelley. *GNUPLOT*. http://www.cs.dartmouth.edu/gnuplot_info.html

[11] Jef Poskanzer. *PBMPLUS*. ftp://ftp.x.org/R5contrib/

[12] John Boswell and Dan Starer. *Five Rings, Six Crises, Seven Dwarfs, and 38 Ways To Win An Argument*. Penguin Books, New York, NY, 1990.

From Something to Nothing (and back)

Gretchen Phillips – University at Buffalo

ABSTRACT

This paper addresses the experience that I have had in making the transition from technician (UNIX systems administrator variety) to management. It discusses some of the issues that I have had to cope with in this unfamiliar territory. If this paper has any management-speak in it, it is only because I have included it by accident, as my background is as a technician. I won't say that *nothing,* but likely *very little,* in this paper has anything to do with management theory; rather it is just the experience that I have had over the past several years.

Introduction – A Brief History

It's been seven years since I presented my first LISA paper (Monterey) and six since my last (Austin), and something has happened to me in the intervening years. Where could six years of my life have gone? Have I done anything interesting in System Administration in this time? Have I done anything interesting in *any* field in this time? I'd like to address this paper to those system administrators who have grown from their technical roots and branched into what from a technician's point of view, appears to be nothing and to those who may one day step into the void.

My daughters visited the office with me last fall and when we went home at the end of the day, they said "Mom, all you do is talk all day. What's your job?" I tried to explain to them that I go to meetings, I talk to people and I read (and sometimes answer) my mail. But after saying this, I realized that I too felt like I didn't really "do" anything anymore. I didn't have xmeters running on my screen watching the machines; I had extracted myself from any mailing list that actually monitors machines; I delayed in learning PERL until nearly every other human on the planet knew more than I did about it; I hadn't written a useful script of any kind in a very long time; I was feeling quite dissatisfied about the amount of work that I would get done in any day; sadly, I had become a manager.

Persisting in this depression much longer than anyone would like, I finally snapped out and realized that not only do I do something; I go to meetings, I talk to people and I read (and sometimes answer) my mail; but that what I do is important and clearly some of the techniques that I'm using are valuable. This revelation occurred after returning to work after being sick (and I couldn't work from home) and finding a huge pile of things that needed attention on my desk and in my mailbox. And, not one of my technicians (system administrators) had attended to these matters. In fact, several projects were waiting for me to help them along to the next step.

Making The Transition

Making the transition from technician to supervisor has been one of the most challenging of my career. I am (IMHO) a good technician and from my annual reviews, my supervisors have agreed. Why was I picked to be the supervisor of a growing group? Well, to be perfectly honest, I was the group. So because of seniority, technical skill and potential (all nonsense in this case), I became the supervisor. Now, several years later, the group has seven members, and I am in the process of hiring four more as I write this paper. We have already started usurping available people from other groups just to keep up with our projects. Two of these will be transferred to my group when the current hiring process is over, giving a grand total of thirteen. It has not been an easy road and supervising twelve people is not at all like supervising one or two.

Who Should Be Boss

Sometimes a supervisor is promoted from within, sometimes they are brought in from the outside. Choosing a supervisor for technicians can be tricky. In my limited experience, I have found that technicians (system administrators) can be a fussy lot. Each one has their own area of specialty. Each one has their own way of doing a job. Each one has their own peculiarity(ies). Supervising a bunch (group to be articulate, horde to be precise) of system administrators can be a challenge. Picking the right supervisor can be equally challenging.

Qualities of a Supervisor: What to look for

I think that there are two levels at which someone should be evaluated when being considered for a position in management. The first level is abstract and includes qualities like:

- the ability to learn and understand the material
- the desire to work hard
- initiative
- creativity
- personal integrity

These qualities set the tone for the work environment.

While exact technical skills may not be necessary, the supervisor must be able to understand both the material and the problems that the technicians will face. I think a supervisor cannot competently assign work if there isn't a clear understanding of the resources, time and expertise required to get the job done.

It has been my experience that I work harder now as a supervisor than I did as a system administrator. (Is that possible?) I'm constantly concerned with all the projects, the big picture and the details of picking up the slack. If I didn't like to work hard, I'd be in the wrong job. Initiative and creativity are related to how projects and problems are approached. Both are necessary for the supervisor to be a leader. Finally, I think that there is no substitute for personal integrity, either inside or outside the work place. If I say that I'm going to do something, I'll do it. If someone else tells me they are going to do something, I expect that they will. If they don't, I'm likely to start avoiding interactions with that person. Supervisors aren't perfect with all of these qualities at all times. I'm not. My supervisor isn't. However, they are a foundation upon which good leadership and supervisory skills are built.

At the second and more practical level, a supervisor must be able to (minimally):

- delegate responsibilities
- accept that a job might not be done the way she might do it herself
- see the big picture
- behave in meetings
- speak in complete sentences

Some of these may seem obvious, but they aren't to everyone. In fact, two of the things on the list may seem like they are a joke. They are not. It is important for a supervisor to be articulate and to have some self-control, thus the *meetings* and *sentences* entries on the list. The job of a supervisor isn't primarily technical. It is important to understand the work that your people do, but most of my time is spent going to meetings, reporting status, writing job descriptions or performance programs or other non-sysadmin stuff. There are some advantages to having a system administrator background before becoming a manager. I do understand the work that my group does. I can fill in when necessary if jobs get dropped on the floor. I use my organizational skills a lot because most of my time is now spent, going to meetings, talking to people, and shuffling paper (or bits).

Becoming a Supervisor: What to work on

Being a good technician doesn't automatically mean that you will be a good supervisor. The list above is a place to start when evaluating your skills. Start with the *big picture* and see if you can see where you fit as a worker into the organization. Start to appreciate the strengths of your co-workers. As a supervisor, you will need to appreciate the technical skills of others, so start to recognize them for what they are. They are not the competition. This is particularly important if you have the opportunity to move up within your organization (rather than changing companies or departments). I've heard it said, *"Be polite to your enemies; some day they may be your friends."*

If you haven't had the opportunity to lead a project, ask for it. Take the lead on a new project and enlist the help of others. If this isn't possible, then start to show your talents in other working groups in which you participate. Ask the project leader for extra assignments. This will help you practice for leading a project of your own. But try not to get stuck doing all the managerial work without ever getting a commensurate recognition. *Campus Network Committee Chair* is the canonical example of work with no recognition. Be careful about taking projects that have no return.

Learn (and use) the basics of spelling and grammar. You don't have to be an expert and know the proper time to use "that" or "which" but do learn the difference between "site" and "sight" and "your" and "you're". Simple things, like complete coherent thoughts, go a long way towards increasing your value as a supervisor. Management is a communication job, so points need to made clearly and with context.

Self-control is an essential *boss* feature. Unfortunately, I never got in line for this quality, so I have had to learn it. Start practicing early. Learn when to speak and when not to. Play nice. Now, I don't mean cave in when pushed or sit silently at the abundant meetings, but do the things of playing nice at work. Be prepared; don't expect someone else to do your work. Be thoughtful; take your time when expressing yourself and think about *how* what you do and say will impact the situation. Be consistent; make sure people know what to expect from you and then give it consistently. Finally, be ready to politely stand your ground.

I have found that having a good model to watch has been particularly useful in learning self-control. You needn't limit yourself to one role model and you can pick one from a variety of places. I learned a lot from my softball coach when he moved me from second base to third for one season. I complained and he said that he was the coach of the team and then gave several reasons, (that I understood and remembered at the time) one of which was for my own self-discipline. Watching how he handles the team gives me a model different than that of watching the various supervisors in my building.

Observing how people respond and to what they respond helps me decide how to handle myself

in different situations. Being generally more observant has been beneficial. Learning to be observant is again a matter of self-discipline. Finally, I still struggle when the people that I'm dealing with don't understand the material and I must repeat or restate or resend things. Often it isn't that they aren't listening, it's that they aren't understanding. In these cases, I need to be much more patient and give them time to absorb.

I'm Your Co-worker and Your Supervisor

Stepping into a supervisory position can be difficult, no matter what the circumstance. If you are hired from outside, the technicians can be justifiably tentative in the new situation. If you are promoted from within, there can be feelings of resentment, instead of good will. Making the transition to supervisor can be particularly difficult after having been one of the group. However, my own situation is neither of these, since there was no group until someone was hired to work with me. In the early days of my little group, the two of us worked together on all the projects, and while I was the supervisor on paper, we were more like a little team. As the group began to grow, and I did less technical work and more paperwork, there were times when my authority was challenged. I was looking at the big picture, all of the projects, while my technicians were primarily looking at their projects. Not only was our group growing, but the role of UNIX in the university was growing and changing. UNIX was becoming mainstream and used not just by the Computer Science Department or School of Engineering. My staff didn't like it when I said something had to be some way that was new or constraining. For example, picking the date to go to Solaris was a real battle. My supervisor wanted it as soon as possible, my staff wanted it never and I wanted it only when it would be reasonable. I must admit that we went forward with some of the staff kicking and screaming.

I Am Your Supervisor

Being the supervisor isn't a role that is inherited, it is a role that is earned. Dealing consistently with staff, remaining technically sharp and getting the job done are all important features of being a supervisor. These are important not only from the point of view of the workers but also from that of the supervisor's supervisor. My supervisor would pass me over in a second if I had a group that wasn't producing. And yet, he doesn't second guess my recommendations because we are getting the job done and we are working within his bounds.

One thing that I think is important in the supervision process is having the authority that goes with the job. Fortunately for me, and my group, my supervisor does permit me to make decisions that directly affect my group. He isn't particularly concerned with which individuals I hire because I am the one that must work with them day to day and supervise them directly. He doesn't particularly care which software solution we might choose as long as it meets the criteria that we define at the outset. This hands-off style allows me the position of really being in charge of my group. In turn, my group looks to me for leadership, rather than looking past me to my supervisor. Sometimes, I must defer to his judgment or wishes, but for the most part, he lays out the projects and we solve them as we see fit.

What's Important

Figuring out what's important in the management role can be a difficult task. I wanted to keep doing the work, and for some period of time this was appropriate, however as my group grew, I had trouble (and sometimes I still do) figuring out what is important for me to do. It turns out that for my particular job, it is very important for me to talk to people. It is important that my technicians know what to do and that our clients know what we are doing. So, given a choice between working on a technical challenge or informing clients or staff about project status, it is more important for me to do the communicating and leave the technical challenge to my staff. Communication skills and communicating activities are important parts of my job.

Letting Go and Hanging On

It was hard for me to let go of my technical work. I am a fussy technician and like work done in a particular way. It was very hard for me to let go and see others do *my* work in a way that was not the way I would do it. Guiding the group, performing a leadership role, while letting go of the implementations is an important balance. If I tried to dictate every detail, my workers would hate me and I would be frustrated because the work still wouldn't be done the way I would like. Still, I can't have them running around doing things however each one pleases. I have found that developing their individual expertise, encouraging their ideas about how projects should be done and keeping them satisfied can be done by keeping my fingers out of their work, just as my supervisor keeps his out of mine. Additionally, when talking with other managers about collaborative projects, if it seems appropriate, I make the effort to mention specific good works of project members (no matter who the supervisor is) as well as mention to my supervisor when they are doing good work. This puts the credit where it belongs. I cannot hang onto credit that is not mine.

Keeping Workers Happy

It is possible to keep workers happy? I wasn't happy about my job as a manager for a long time because I didn't understand my role. Technicians can suffer from this same problem if they do not understand what their job is and how they fit in, and

I don't just mean performance programs. As a manager, it is important to understand the dynamics of the group of people you work with.

What Really Makes Happy Workers?

There are some simple basics about what make people happy at work. In no particular order, they include:

- Compensation
- Flexibility
- Recognition
- Getting jobs done
- Satisfactory working conditions (even better: excellent conditions)
- Interesting projects

I work for a state organization and compensation can be a serious problem. Salary figures are below industry for similar positions and direct benefits are usually satisfactory but not necessarily outstanding. I have found that making sure that my workers have satisfactory equipment to use from home (at office expense, not theirs), workstations, modems, office phone by-pass systems, travel to conferences and the like, is fundamental to making their work environment pleasant. Additionally, offering them the flexibility to work from home improves their working conditions and in turn their mood about the job. Providing simple things like printers in office areas, rather than only at the central print site, makes their work more efficient and makes them feel that their time and work is important. (It is.)

Working conditions are an important part of the total workplace. Having a chair that fits, a table that's right and lighting that is agreeable are all simple but important ways to make the workplace more pleasant. Lighting and eye strain problems are pervasive in the workplace. Allowing each worker to customize their lighting to satisfy their personal needs can have great rewards. The workplace isn't a one-size-fits-all place.

Each worker is has different recognition needs. Some are satisfied to work for a job well done and don't need special encouragement, while others need to be recognized with a good word. It doesn't cost me anything to say, "Thank you," when someone completes a job. If I get in the habit of appreciating and acknowledging the work of each of my technicians, it then doesn't matter who needs acknowledgment and who doesn't care about it, because everyone's contribution is being acknowledged.

Finally, fitting the available work to the available staff can be a challenge too. In the long run, if technicians feel that their projects are stimulating, then I think, for the most part, things will be OK. Of course repetitious jobs are boring, but some people like that. Some really like the end user part of the support, some like the installation, others like the problem solving. Making sure that every one has some projects that they find truly interesting is important. Again, the workplace isn't one-size-fits-all.

What Makes The Boss Happy (and Productive)?

I find myself being surprised at the things that now make me happy at work. As a student in Computer Science, and later as a system administrator, I worked on projects for the adrenaline flow that came with accomplishing a task, finishing a program, or having a user finally go away happy. As a manager, I rarely get to have one of these encounters, so I must find other sources of satisfaction.

I particularly enjoy the hiring process: the hunt for new technicians, the thrill of finding good ones. (More on this later.) I am finding satisfaction in the growth of my group; in the apparent confidence that my supervisor has in my management ability. These types of people interactions aren't something that I previously thought I would enjoy.

Getting Things Done

What does it mean to get something done? I know for sure that I am not a 100% type. However, I can't decide if I'm a 95% or a 105% type. By this I mean, some projects get to be 95% done and that's good enough and I say leave them. Others are finished and yet I want them improved, changed, or done to the final detail. As a technician, I was more in the 105% category. As a manager, I cannot demand 105% from my technicians, nor from myself because sometimes some parts of a project are out of my control. So, we just do the best we can and go to the next thing. As a manager, it is my job to say when a job is finished or not.

The Big Picture

Part of any management job is looking at the big picture; seeing all of the tasks that all of the technicians do. Having some idea about how they should be assigned and prioritized is important. Additionally, it is important to know when a project is complete enough, relative to other projects and priorities, that it can be closed, left behind or ignored.

I get a feel for the big picture based on many things. These include: where we are in the academic year; where we are in the fiscal year; how much money we have left; what projects we currently have active; client resources for projects that need my staff; and of course, who wants to take a vacation and for how long. There is a certain cycle to our work based on the academic calendar; we have fixed deadlines that can't slip for some projects while others can slide weeks and sometimes months. Deciding what priority any particular project should have is a mixture of the resources and deadlines along with some amount of gut feeling when no clear priority exists.

The Details

Technicians are often detail driven. (I am.) Just one more feature, one more bug worked out, one more install finished. Details can prevent the big picture from being seen; the old forest and trees problem. It is the manager's job to decide which details are important (need attention) and which ones are the last 5% that can be ignored. This is a skill that I've had to learn. It has not come naturally.

If a project is stalled, perhaps waiting on a controller or disk, I may be inclined to ignore the detail of price and say that we should pay whatever price is necessary to keep a project rolling. Under other less critical circumstances, I may ask more than one staff member to shop around, looking for the best price on hardware configurations. The detail of price sometimes can, and must, be ignored. The details ignored on any project are most often related to the deadlines of the project. Finally, sometimes details aren't ignored, they are simply deferred until resources are available.

Preparing For New Duties

Some of duties of a manager are different than those of a system administrator, and yet some of them are similar. Organizational skills are still important, just different things being organized. Users don't go away totally, their faces just change. My method of preparation has been to use my intuition in areas where I think I have some and try to read up on areas where I don't. Formal education is always a possibility but not a particularly simple one. If you decide that you want to go back to school and study management, be prepared to do a lot of work.

There are many facets of the management role. The following sections give a brief overview of the kinds of things that I've encountered.

Hiring Process

Hiring for the State University of New York is a complicated process: papers filled out in a certain way, job description and postings of only 6 lines, review panels and the like. Finding a suitable set of candidates is one problem, picking one is another. Evaluating resumes turns out to be one of my favorite tasks. Some of the candidates are obviously inappropriate but as the field gets narrowed down, the adrenaline begins to build. I look forward to meeting these people, finding out what they really know and selecting one to work in our group.

Doing the initial phone interviews is one of my least favorite parts of the hiring process. Speaking to someone who may be my potential prize, trying to determine if they are worth a personal interview is one of the hardest parts for me. The phone interview is a necessary part of the process though. A 15-20 minute phone interview can tell me if the candidate has potential since I don't have time to schedule personal interviews with all candidates who might be minimally qualified.

Preparation for the personal interview can be as much work for the interviewer as for the interviewee. Figuring out what kinds of skills a candidate has and if those skills fit appropriately in your group can be a challenge. I try to focus on problem solving and ability to learn new material, rather than on specific facts known.

In my most recent personal interview, the candidate asked me, "Don't you have any more questions for me?" I think he was genuinely surprised that I didn't ask many questions with right or wrong answers but focused more on his problem solving skills. Perhaps he wanted to show off some obscure tidbit that he had recently acquired but apparently he didn't get the chance.

Performance Programs and Evaluations

Depending on your organization, performance programs and evaluations may be either a big or small part of your job. I've found that using performance programs that are similar is an effective way of cutting the amount of time that I must spend preparing them. Customizations come in the details but the layout and tone are all the same. If your organization provides some template or form, by all means use it or make one of your own.

Using evaluations effectively is another challenge for me. I want to use them primarily for a big praise of my technicians. It is often the only time the upper bosses ever see or hear the names of my people. I also try to make effective use of them for recognizing areas of improvement and suggesting other items where improvements are possible. Wording these can be a tricky task but I think worth it in the end.

Meetings

I go to meetings.

I was unprepared for the amount of time that was consumed by going to meetings. It seemed for a while as if I spent most of my week with my notebook, going from conference room to conference room, office to office, meeting with people. As time passes, I am trying to teach others that information can be passed without face to face meetings. Although some days still seem as if I spend the entire day with a parade through my office or on the meeting circuit.

With two groups, we have gone from weekly meetings to monthly updates with actual work happening on-line. One of these groups was a mixture of managers at my level and technicians from our groups and the other was of technicians from several different groups. This has not been an easy task but meetings are not my favorite thing so I've made an effort to encourage people to meet less and work more.

The New You (Politics In Disguise)

Most anyone will tell you that all work places are political entities. They are both formally and informally structured. People have supervisors and workers and there are unsaid political organisms also. I (like many others) prefer to keep as far out of any overt politics as possible, however there are some practical things that I have learned in the past several years.

The thing that it has taken me the longest to learn, and I am still learning it daily, is to keep from stating my opinions as facts. The technician part of me can see a clear path from A to B or for solving problem Q but my clients don't have that mindset. They want to know the options, and ultimately want to make the final decision. Helping clients make the *right* choice is one of my new developing skills. This means laying out options and consequences and making sure that they fully understand the implications.

Another simple maneuver is to refrain from expressing anything with the word *you* in it. This keeps clients from feeling that they must be on the defensive. Notice the difference in tone between:

- If you don't back up the disks and there is a failure, you will lose data.
- If the disks are not backed up and there is a failure, data will be lost.

It's a matter of rereading everything that I write and a small price to pay for the good will that it creates.

Keeping Technical Skills Sharp

Keeping my technical skills at a reasonable level continues to be a difficult task. My primary difficulty here is finding the time to keep up. I depend heavily on my staff to help me keep up to date. I often find myself asking, *Where do we keep ...?* or *How did you ...?*, but my staff doesn't appear to mind telling me as long as I don't repeatedly ask. Also, this gives them an opportunity to talk about what they've done and demonstrates that I think that it was interesting enough to ask about. (I think that as long as I don't act like a user, I'll be OK.)

I gave up reading the trade rags and now just glance through them when I see something interesting. Again I am depending on my staff to let me know about things that are not obviously thrilling but perhaps interesting in our environment.

Is Management Vocabulary A Waste Of Time?

Part of my so called management training was an invitation to Service Excellence Assemblies. I carefully avoided attending these meetings but there was a lot of chatter among other managers about thinking up projects that would provide examples of service excellence. I was coerced by my then-supervisor into writing up one of my then-current projects and submitting it for consideration for a Service Excellence Award. I got the award. In my opinion, service excellence isn't a matter of contrived projects; it is a matter of excellent service.

The concepts that are the foundation of the buzz words of management, like diversity and team building, only became clear when I experienced or really observed them in their natural forms. Contrived team building is a waste of time. Setting people to work together on a common goal with shared responsibility and due recognition is a worthwhile project. I genuinely think that the vocabulary makes no sense until there is some concrete context and even then it still sometimes appears artificial. This isn't to say that the vocabulary is a complete waste of time, simply that the vocabulary should be neither the means nor the end, it should be the descriptor.

Diversity

Some people might say that my group is diverse because we have a rich racial mix, or because we have 4 men and 3 women or because we come from a wide variety of social backgrounds. While these things may be building blocks of diversity, I have found that in fact our group is in some ways quite homogeneous. We are all in the 25-40 age range, we all consider our work our hobby and are amazed that someone pays us to do it, and every one has that just-right smattering of compulsive personality. Our work habits are about the same, as are our hours and one person can, for the most part, pick up work that another has dropped with little difficulty.

Yet, there is diversity in the group. I learned this lesson the hard way when I jokingly told one of my young members to lose his tie. He dutifully left it at home but I later realized that he was more comfortable wearing it since he was working with administration personnel and all of his clients were wearing them. They were expecting him to meet their dress code and I was expecting him to meet mine (I foolishly thought I didn't have one).

Another area of diversity is learning styles. I have some technicians who just dive in and try things out, others read the manuals, while still others learn best with formal classroom training. It is important for me to provide them learning environments that fit their learning styles. It is equally important for them to understand that it isn't appropriate to send everyone to the same kinds of training. Related to this is assigning projects that work with learning styles. Troubleshooters need to be the type that dive in and figure things out. End user support might better be done by those that love to read the manuals.

Finally, I have people in my group who love the end user part of system administration. I have others that hate it. I have one worker who will gladly pitch in on any project and the others find it comforting to know that he is available to help if they need him. He isn't my best technician but I

wouldn't trade him for the world because he adds stability and kindness to our group.

Team Building/TQM

I will gladly lump Team Building and TQM into the same worthless vocabulary box. Applying artificial mechanisms on the top of a broken management technique doesn't work. I worked for a man who tried this. I can attest from the bottom that it doesn't work so I wouldn't try it from the middle or top. However, the underlying theory of having workers not only feel responsible but *be* responsible for their situation does work.

If I have achieved this at all, it has been by allowing my technicians the opportunity to voice their interest in specific projects and then by giving them both the responsibility and the credit for projects well done. If things appear to be going not exactly right, I'll ask if they need anything, resources, help, or whatever. Often, knowing that I'm available with resources is enough to help them through. Everyone realizes that we draw from the same pool of resources so if they are short this time and call out extra resources, they know that they'll be called upon in the future when someone else is short. Actually, the group is now large enough that there are some subgroups beginning to form where they independently help one another out on stressful projects.

Time Management

About two years ago, I was assigned the task of surveying all of Technical Services to see what skills the technicians would like to improve. Several said they needed help with time management. (My group didn't answer me at all, I guess that they didn't have time.) I have no key to solving the problem of too many projects and customers and not enough resources to handle them all.

One relief that I do offer my staff is that if they are having trouble prioritizing their tasks, they can feel free to ask me to do it for them and I will then take any heat that they get as a result. They have found this to be a workable solution because they can spend their time working on projects, rather than explaining to any particular customer why that project isn't being done at the moment. Basically, I run interference for them but it does free up their valuable technical time to get projects done.

Time management for myself is a different problem as I spend way too much time in meetings and talking on the phone. But, as I have realized over the last several months, this is in fact what I'm being paid to do.

Things I still hate

I sometimes think that I would be much happier if I were still exclusively a technician. Some of the parts of being a manager really do not appeal to me. Yet, I am aware that I would be much less happy if someone else were trying to tell me what to do, or leading projects in a way that I didn't like. As a result, I do things I don't like.

Performance Programs and Appraisals

I understand the need for technicians (and even managers) to know what their jobs are and to be able to know what is expected of them. I also understand that it's good to know how you are doing on your projects. The bureaucratic organization that I work in makes it almost impossible for me to reward good performance and yet I am required to annually review the performance of all my technicians. My group will soon expand from 6 direct reports to 12. This means more paperwork than I will ever want to do in my entire life.

Time Wasters

Finally, the thing I dislike the most are those projects, meetings or people that waste time. It frustrates me to no end when people say they want to have a meeting about some nonsense thing or they want their Graduate Assistant to come an interview me about my input on some project that has no bearing on my group or our work. I am a technician at heart. I'm goal-oriented and like to see concrete results. I like to see projects completed, be they machine installations or finishing the paperwork for a new hire. To this end, I try, as best as possible, to avoid things that don't give me that satisfaction.

Conclusion

While this paper does not cover the traditional areas of System Administration, it does present the experience of a technician turned supervisor. I hope that it has been of value to those who are supervisors, those who want to be supervisors and those who aren't and don't want to be. For that last group, perhaps understanding better how to get the best out of their boss will help them be more effective in their jobs.

Is there conclusion?

I by no means feel that I know all the issues related to management of technicians. I've only just recently learned that management has content. There are many things for me still to learn. Also, my management environment has changed frequently. I have had three different supervisors, worked in two different divisions of CIT, and had my group grow from just me to 12 full time direct reports. I don't expect there to be a time where I think, *This is it. I've arrived.* There are always new things to learn, new projects to tackle and improvements to be made.

However, I do feel that despite all my idealism (and perhaps naivete), that there is no substitute for treating people kindly, caring about their needs and making their work experience as pleasant as possible. This is, after all, how I like to be treated by my supervisor.

Acknowledgments

This paper would not have been possible if, somewhere along the way, someone hadn't thought that I had supervisory potential. Thank you to that someone. Thanks to my supervisor, Chuck Dunn, for giving me the work environment that I have and for thinking it was a good idea to write this paper. Also, thanks to Lisa, Paul, Daniel, Zach, An-Tzu, and Andy for putting up with me daily. And finally, thank you to Dave Carr, John Quarterman and Gail Rein for reading this paper while in progress. Everyone should be so fortunate as to have friends like you. Thank you.

Author Information

Gretchen Phillips is the Manager of the UNIX Support Group in Technical Services at the University at Buffalo. She has been working in UNIX System Administration since 1983. She can be reached at gretchen@acsu.buffalo.edu

References

Schoenberg, Robert T.; *The Art of Being a Boss* Harper & Row 1978

Shurter, Robert L.; *Effective Letters in Business* The Gregg Publishing Company 1948

Tarshis, Barry; *Grammar for Smart People* Pocket Books 1992

Yate, Martin; *Hiring the Best* Bob Adams Inc. 1993

Metrics for Management

Christine Hogan – Synopsys, Inc.

ABSTRACT

We were recently asked by management to produce an interactive metric. This metric should somehow measure the performance of the machine as the user perceives it, or the interactive response time of a machine. The metric could then be used to identify unusual behavior, or machines with potential performance problems in our network. This paper describes firstly how we set about trying to pin down such an intangible quality of the system and how we produced graphs that satisfied management requirements.

We also discuss how further use can be made of the metric results to provide data for dealing with user reported interactive response problems. Finally, we relate why this metric is not the tool for analyzing system performance that it may superficially appear to be.

Introduction

Metrics are being used to produce charts and graphs of many areas of our work. For instance, at Synopsys, the number of calls opened and closed in a given day or month are charted, as are the number resolved within a given time period. There are also metrics that monitor more specific areas of our work, such as new hire installs, and more general things, such as customer satisfaction ratings. More metrics are constantly being devised by management, and put into place by the systems staff. One such metric was the "interactive metric", which was intended to measure the interactive response time of a machine and highlight any problems that might require further investigation.

While it is possible to monitor a number of different components of the system that could influence system response time [1, 2, 3], there are currently no tools available to monitor the performance of the system as the user perceives it. We were asked to develop such a tool. The interactive metric was meant to imitate a user as closely as possible, and measure how long it took to perform a "typical" action.

In this paper, we first provide some background information on the particular installation in which this metric was to be installed and describe some of the issues that arose in its design. Then we present the actual design of the metric. Thereafter we focus on the issues that arose in interpreting and presenting the data that the metric gathered. These issues, in fact, were the key ones when it came to presenting the results to management and determining to what extent we, as system administrators, could find them useful. The paper concludes with a discussion of the limitations of the metric, in particular from a system administrator's perspective, and describes some possible extensions that could enhance its current utility.

Background

In this section we present a description of the site at which the metric is running. Also included are some tables that relate machine names to architectures, fileservers, subnets and the role of the machine. This data provides some insight into the graphs produced later in the paper. The motivation and intentions behind the development of the metric are also briefly mentioned.

The Site

The site at which this metric was developed is in Mountain View, California, in the head office of a company with a number of sales offices throughout the US, Europe and Asia. Most of these sales offices are on the company WAN. The Mountain View campus network has a services backbone, to which the shared servers are connected [4]. The servers are often dual-homed with the other interface being on a departmental subnet. Each department has its own subnet, with a number of fileservers and compute servers, as well as desktop machines. Desktop machines are a mixture of X-terminals and workstations. The workstations generally have local swap, remote root, and shared, read-only usr partitions. There are also a large number of Macintoshes and PCs in this site. However, the metric is not run on those machines.

The metric was initially tested on a small subset of machines on a few of the subnets. For each subnet that was included in the testing phase a fileserver was selected, along with a number of machines that were known or suspected to be slow, and machines that represented some cross-section of the architectures at the site. The fileserver that was selected for a given subnet was one of the fileservers typically used by the machines on that subnet. Table 1 groups the testing machines into subnets and

shows the fileservers used, while Table 2 gives the architecture of the fileserver.

During the initial phases of testing and designing the metric it was also run in a number of WAN-connected sales offices, which are not shown in Tables 1 and 2. The sales machines were dropped since the complaints that were being received from people in the sales offices related to WAN latency issues, and the metric is not a suitable tool for monitoring or graphing the performance of a WAN connection.

Motivation

The idea for such a metric was arose from the observation that sometimes a user will complain that the system is slow, or the network is down. There was a suspicion that "the network was slow" was becoming the modern, hi-tech, equivalent of "the dog ate my homework". However, without any form of metric or supporting data, it was impossible to try to refute those statements, or defend the state of the system/network as a whole. Therefore, there was a desire to try to quantify the experience of the "average" user, and to have some form of data to use for the basis for discussion.

An analogy that is frequently used at Synopsys[1] is the comparison of a network with the freeway system in Los Angeles. At any one time the network, or freeway system, is neither up nor down. Some segments of it may be down, or extremely congested, but others will be in perfect working order. In fact, this analogy extends to the metric. The metric is the equivalent of taking sample trips along certain routes at different times during the day, and measuring the time taken, to get a feeling for "delay". We recently discovered that such sampling is one of the methods actually employed by civil engineers in the study of traffic routes and delays.

The idea behind the metric was to develop a tool to measure "performance" at a high level, as the user sees it, rather than breaking the system down into a series of components, none of which in itself means anything to the user. The metric was not intended to be an all-purpose instant system and network analysis tool for the system administrator. However, we, the system administration staff,

[1]The inspiration for which was Eric Berglund's, with extensions from Arnold de Leon.

Machine	Subnet	Fileserver	Architecture	Purpose
mingus	72	anachronism	Sun 4/40	Desktop
gaea	64	anachronism	Solbourne S4100dx	Many services
underdog	72	anachronism	Sun SS4	Desktop
kency	72	anachronism	Sun SS10	X-terminal server
amnesia	100	anachronism	Sun SS20	NAC administration
fili	100	dempsey	Sun SS10	CPU server
mahogany	74	dempsey	Sun IPX	Build machine
redwood	74	dempsey	Sun IPX	Build machine
mordor	124	dempsey	Sun SS10/512	Porting
canary	68	anachronism	Sun SS4	Desktop
paris	68	mammoth	Sun 4/40	Desktop
goofus2	68	mammoth	Sun 4/50	Desktop
orac	68	mammoth	Sun 4/40	Desktop
millstone	92	almanac	SS20/61	Xterm Server
droid	92	almanac	SS10	LISP testing
mercury	92	almanac	Sun 4/60	Desktop
sark-92	92	almanac	Sun SS20	CPU server
jose	92	almanac	Sun 4/60	Desktop

Table 1: Subnet numbers and fileservers of tested machines

Server	Subnets	Architecture	Users
anachronism	72, 100	Network Appliance	NCS
dempsey	100, 116	Auspex NS6000	Porting Center
almanac	92, 100	Network Appliance	Design Verification
mammoth	98, 100	Solbourne S6/904	Product Engineering

Table 2: Architecture of the fileservers

thought it would be nice if it did give us some useful feedback in return for the time and effort expended. We will examine to what extent our goals and our hopes were met.

Designing the Metric

There were a number of issues involved in designing the metric. Not only did it necessitate deciding upon a typical set of actions that would be sufficiently non-intrusive, but it also involved determining a typical environment in which these commands would be run. Of particular interest were the differences in performance for different groups of users, who were on distinct subnets and using separate fileservers.

Design Issues

User perceived system performance cannot be simply measured by a single number. A user's view of the performance of a system is formed on the basis of how long it takes to perform a particular job, and thus two different users on the same system may have differing views on the performance of that system [1]. Thus, an essential part of designing a metric that would be of some use to us involved determining what a typical user's job involved. Since a significant portion of our user community is involved in software development, it was decided that the typical action that we would model would be the edit, compile, run cycle. While this series of actions is not typical of all sections of our user community, the only way that we could make comparisons between different networks and fileservers was by running the same metric on all groups of machines. Thus, this set of actions was decided upon, while acknowledging that it limited the usefulness of the results for some sections of the community.

The first phase also involved performing some sanity checks on the metric using data gathered from an initial set of metric runs. This initial examination of the data was intended to verify that the metric was producing at least superficially believable results. It also enabled us to alter it somewhat to emphasize any differences in performance that existed.

Simulating an Interactive Session

To simulate an interactive session, we first tried using Dan Bernstein's `pty` package [5] to both provide ptys for `vi` and to simulate the speed at which a user types. While the results showed some variation from machine to machine, it was not significant, and we did not feel that it represented the differences that we perceived when using the machines. Partial results from this run are shown in Table 3. The metric performed two edits, a `make` of a small project and a short execution run. The results here are correlated with the tables from the "Background" section, and it is noted these results provided a sanity check on the metric by displaying longer times for machines that seemed to give poor interactive response. Even ignoring the results produced by gaea, which suffered from some problems during that week, the relative ordering of the other machines at least made sense.

	Total	Edit 1	Edit 2
kencyr	0.837238	0.413107	0.419813
gaea	2.391545	1.100123	1.275356
mingus	1.064980	0.500288	0.563270
canary	0.525883	0.257806	0.267521
amnesia	0.417002	0.200812	0.215840
orac	1.377794	0.716232	0.660213

Table 4: Initial results from the metric using `chat2.pl`

While the relative performances made sense, the absolute differences in the numbers did not. These machines had been selected to show a wide variation. It seemed that the time was dominated by the speed that the `pty` program was playing back keystrokes, and that actually simulating user input was therefore probably not what we wanted, if we were looking for widely dispersed numbers. The metric was altered to use Randal Schwartz's `chat2.pl` package [6] to simulate user interaction. This method of providing input to interactive programs can send the input to the program as quickly as possible, without simulating a delay between keystrokes. Results from an initial run with this altered metric are shown in Table 4. This initial

	Total	Edit 1	Edit 2	Compile	Run
mingus	53.090136	28.652537	6.077702	18.288256	0.059256
gaea	373.525000	332.572501	7.661372	33.211638	0.067266
underdog	40.528716	28.717238	5.917166	5.866199	0.029712
kencyr	43.236301	28.745429	5.941636	8.513442	0.034764
amnesia	38.046358	28.390941	6.440619	3.173970	0.033065
canary	39.296980	28.499463	5.820487	4.952673	0.021444
orac	52.123264	28.618801	6.015613	17.426540	0.067794

Table 3: Initial results of the metric, using `pty`

run was executed in a different week to that in Table 3, but measured the same sequence of commands. These results show a much clearer separation between the machines, and were considered to be more representative of the differences between the machines. Thus the metric was altered to use `chat2.pl` instead of `pty`, in order to emphasize the differences between the machines.

The Execution Framework

This section briefly describes how, based on the configuration described in the "Background" section, we set the metric up to run on a number of subnets, using fileservers that we could argue were typically used by people in the group to which each subnet belonged.

The metric is currently being run on each of ten subnets that correspond to the primary business units of the company in Mountain View. For each subnet a number of machines were selected to give us a sample of about three machines of each class of machines present on that subnet. For example, if a subnet had Sun SPARCstation 20s (SS20s) that act as compute servers, Sun SPARCstation 10s (SS10s) that act as X-terminal servers, Sun SPARCstation 5s (SS5s) that are on desktops, Sun SPARCstation IPCs (IPCs) that are on desktops, and SS20s that act as servers for the desktop workstations, there would be five classes of systems on that network, and a sample of each class would be selected. In addition, machines that we expected to be especially heavily or lightly loaded were selected.

Each subnet is used by a particular business unit, which uses a number of fileservers. One of these fileservers contains home directories, whereas work areas containing product can be housed on a number of different fileservers. Thus the decision was made that the fileserver that would be used by the metric on a given subnet would be the one that contains the home directories of the people on that subnet. The metric therefore may not in fact be using the machine that a given user accesses while working on a product. However, it is guaranteed to access a fileserver that every user in the group accesses a number of times during the day. Therefore the chosen fileserver can be said to influence the users' perceptions of system performance, in some way.

Analyzing the Results

In "Interactive Session" section, we showed the results as running averages over the course of a week. While this was useful in the initial stages of attempting to produce a metric that gave believable results with significant variations, it yields no further information. The next stage was to produce graphs that were satisfactory from a management perspective, which involved considerable dialogue and a number of revisions in what graphs were produced. We thereafter examined the data produced to see if we could extract any useful information from the users' or the system administrators' perspectives. We discovered that all three goals were quite distinct from each other and involved taking different views of the same data.

Graphs for Management

Initially we produced some graphs that showed the average results on an hour-by-hour basis for each of the internal groups that we were monitoring, where each group was represented by a set of machines on the same subnet, using the same fileserver. Along with these graphs, we also

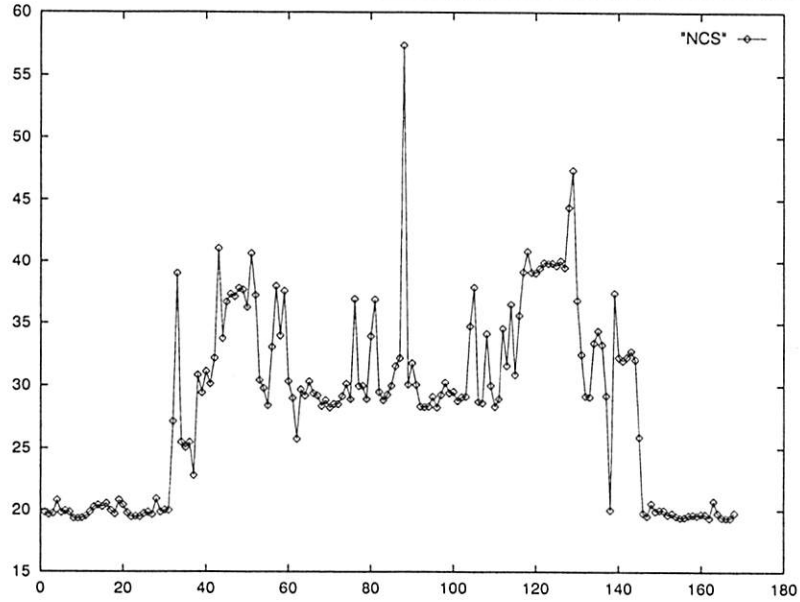


Figure 1: Average results for NCS

produced individual graphs for each of the three best and three worst machines. For example, the graph produced for the Network and Computer Services (NCS) group is shown in Figure 1, and the results from the machine jose are shown in Figure 2. The numbers on the horizontal axis are hours since 0:00 on Sunday, and each graph represents a week. The numbers on the vertical axis represent the length of real-world time that it took to run the metric, in seconds.

Other than noting that the systems in NCS are lightly loaded at weekends, we also noticed that there was generally a trend in the values produced on each machine. What management were interested in were variations from the trend. Since trends are machine-based, we normalized the results from each machine, and plotted the set of machines that were being monitored in a particular group on the same graph.[2] These graphs represent deviations of the machines from their normal behavior.

Another representation of the data that was perceived to be interesting was a stock-market style high, low and average chart, on a day by day basis, based on the normalized results, as described above. This form of graph would highlight machines with a large variation in performance, which could be indicative of a problem.

Relating to the Users

The metric is also being used to gather trend data for a given architecture on a given network. It is thought that this data will be useful as a reference point when discussing performance issues with a user. It should be possible to monitor a machine about which a complaint has been received, compare the data it generates with that of other machines of its class on that subnet, and either state that there is a problem with it, or that it seems to behaving normally for a machine of its class.

Information for the System Administrator

One way of representing the data was to produce a graph for each group that compared the different classes of machines in that group against each other. As expected, the SS20s were shown to be considerably faster than the SS5s or IPXs in these graphs. However, we were able to make a couple of interesting observations from these graphs. Firstly, our SS10s that are used as X-terminal servers yield worse performance than the SS5s that are on people's desks. We also noted that there seemed to be a base "best" performance for each class of machines, which seems to be primarily dominated by NFS response time, but is also a function of the class of machine. This "best" performance time was universal across our subnets with the exception of the Product Engineering network, on which we were using an Auspex[3] file server, rather than a Network Appliance[4] file server. The machines using the Auspex exhibited marginally worse results across the board for this metric. However, these were brief, once-off observations. Of more use and interest would be the use of the metric as a diagnostic or analytical tool.

[2]Due to the intentional clustering of the data around a single area – the "normal" line – these graphs are best viewed in color.

[3]HP IV processor; Storage processor III; File processor II; old-style I/O processor; running 1.6.1M1

[4]FAServer 1400, running 2.1.3

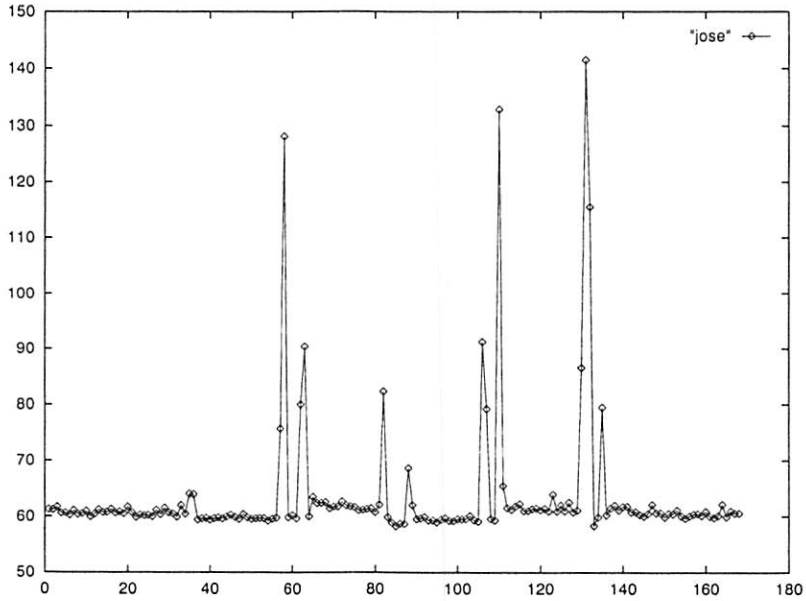


Figure 2: Average results for jose

It has been said [3] that all system performance issues are basically resource contention issues and that the biggest challenge is figuring out which kernel subsystem is really in trouble. In a large site, however, the primary challenge is discovering which individual servers or networks are suffering performance trouble. System administrators may never log into a user's desktop machine. This machine may, however, be central to the user's perception of system performance.

The graphs that are produced for management from this metric do not aid the system administrator in the task of identifying individual machines that have problems because they are based on the usual behavior of each given machine, with no consideration given to absolute performance values. Unusual behavior may, however, be detected from graphs relating the performance of one machine to others of its type, such as the graphs that are being produced for generating concrete data for dialogue between users and system administrators. The utility of both of these sets of graphs for system administrators will be discussed further in next section.

Another potential use of the metric for system administrators is in the area of performance tuning and reconfiguration. Machines at a large site, such as Synopsys, tend to be clones of a standard model. Before changing that hardware model, some testing and benchmarking is performed. This principle can also be applied to performance tuning. A single system could be reconfigured and the performance of the new configuration monitored over a period, with the results being compared with its previous results and those of other machines in that class during the same time period. This data could then be used to justify a decision on whether or not to similarly reconfigure the other machines in that class.

Limitations

While the metric has produced some information that management found interesting, and other information that can be useful in dealing with performance issues that can arise with users, we feel it is of limited use, as it stands, in producing information that a system administrator can use.

The metric can currently be used to detect abnormal behavior of a machine. However, if all of the machines of a particular type are identically mis-configured, and yielding less than the performance of which they are capable, that will not be detected. Even if abnormally bad performance is detected, the metric gives no indication of how the performance of the system may be improved. Also, the numbers that it produces cannot be used directly to say that a given machine is performing well or badly. The metric must be calibrated within the particular environment before the results are interpreted. Grouping machines into classes, and comparing the resuits of a given machine to those of others in its class during the same time period is a step in the right direction. However, there are many other things that can affect the performance of a system which are not taken into account in these graphs. For example, this metric does not indicate the amount of memory or swap space in a system; it gives no indication of whether a machine was otherwise active or idle at the time; and there is not, currently, any way of correlating the results to network load, or NFS performance of the server during the given time period.

Another data-point that we found interesting was that the SS4s produced better results for the metric than SS5s. We use generally SS5s rather than SS4s at our site as desktop machines, because it was felt that the lack of expansion possibilities, and the slightly different operating system outweighed any advantages that the SS4s might have. Thus, this result demonstrates that better performance is not everything when it comes to choosing a machine for the standard desktop model.

One area that we feel is a major limitation of the metric is the lack of any way test X performance, which is fundamental to a user's perception of system performance in our environment. Another correspondingly significant limitation is the inability to run this metric on our other platforms, such as Macintoshes and PCs.

Extensions

The metric could be extended in a number of ways to provide more information for the systems administrator. In considering possible extensions, we consider the reasons behind the aforementioned limitations. We also consider the approaches taken in monitoring and tuning the performance of a small number of systems. We then propose ways of extending the metric to provide additional information without adding unduly to the logging overhead.

Finding the Source of a Problem

While it can be argued that this metric aids the system administrator in locating machines in a large network that may be suffering from performance problems, the graphs represent information at too high a level to even give a feel for what component of the system might be at fault. Tracking down a performance problem on a single machine is discussed in the literature [1, 3]. Automating the testing that would be performed on a sick system could be incorporated into the metric. However, the inclusion of the results of this testing into the logs would result in more data than could reasonably be stored without filling up the filesystem. On the other hand, it is possible to set thresholds in the metric, so that if the time elapsed from the start of a run to the end (or any, clearly defined, intermediate point) is greater than the threshold, the additional information is logged. This form of additional logging could be

a useful extension to the metric, because it would give some context for the spikes on the graph.

We could also consider the relationships between this metric and other tools that monitor the performance of a particular aspect of a machine or network, such as NFS [7, 8, 2]. We would like to consider how the results of these tools could be correlated with those of the metric to demonstrate what influence that aspect is having on the overall user-perceived performance. We feel that this would be a useful and interesting extension, but it has not been implemented as yet.

Real-time Detection of Problems

An extension that was proposed within Synopsys was that notification of a problem could be sent to the administrator of that machine by email or pager, in real-time. This notification would allow the administrators to monitor the state of the machines as the problems were occurring. In addition, it was proposed that notification could be sent to the administrators when the script failed to complete for one reason or another. The most common cause of failure is lack of disk space on the fileserver that a given machine is using, or the server going down. I experimented briefly with automatic notification, and discovered that with even a fraction of the machines that are in the current operational run, the amount of notification is so large as to be overwhelming. I am of the opinion that this is not a useful extension, as it stands.

It could possibly be made useful, however, through employing `syslog` or by linking the metric in to a real-time monitoring system such as Netlabs. We have not experimented in this direction, but either of these approaches would conquer the flooding effect.

Using Alternate Sequences

It was mentioned in "Design Issues" section that the user action that we model is the edit, compile, run, cycle. It has also been suggested that we may want to monitor file transfers over the various WAN connections, or perhaps the amount of time it takes to perform a particular sequence of queries on one of our databases.

The metric is written in such a way that, providing you are comfortable with `perl 5` and `chat2.pl`, it is trivial to drop in any sequence of transactions that could be performed by a user at a standard shell prompt. It does not support timing X applications, however.

Conclusions

The metric produces some useful ways of visually demonstrating normal and abnormal behavior of machines in a network to management and the user community. It gives a high-level, generalized overview of how the performance of the machines, or "network" in a particular business unit is performing. The approach taken is similar to the civil engineering approach of taking sample trips to measure delay on particular routes, or in a particular network of roads.

The metric does not, however, produce data that in itself is useful to system administrators in finding trouble spots in a large network of machines. It could be extended to be somewhat more useful in this regard, but it would take a considerable amount of work. The amount of time that would be spent extending the metric in this way must be balanced against the usefulness of the results and the intentions behind running the metric. In our case, we decided that it was not worth the effort since the metric was not intended to produce information that would be directly acted upon by the system administration staff, but rather to provide a general overview of the performance of the network as a whole.

Acknowledgments

The encouragement and advice that I received from Aoife and Paul were instrumental in the production of this paper, and for that I am profoundly grateful. I owe thanks, and perhaps a few drinks, pizza, or something interesting to read, to my proofreaders, Aoife, Jeff and Becky. My thanks, also, to all the unwitting participants of my experiments – the user community of Synopsys, Inc. – for their patience in putting up with the additional load on their machines. And equally, to Eric who believed in the metric, and to Randy whose fault it was in the first place. Finally, not forgetting Arnold, who not only proof-read the paper and avoided implementing the metric, but also takes some of the blame for inventing it in the first place.

Author Information

Christine Hogan is the security officer at Synopsys, Inc., in Mountain View, California. She holds a B.A. in Mathematics and an M.Sc. in Computer Science, in the area of Distributed Systems, from Trinity College Dublin, Ireland. She has worked as a system administrator for six years, primarily in Ireland and Italy. She can be reached via electronic mail as chogan@maths.tcd.ie .

References

[1] Mike Loukides. *System Performance Tuning.* Nutshell. O'Reilly and Associates, Inc., 1990.

[2] Gary L. Schaps and Peter Bishop. A Practical Approach to NFS Response Time Monitoring. In *Proceedings of the 7th System Administration Conference (LISA VII).* USENIX, November 1993.

[3] Marc Staveley. Performance Monitoring and Tuning. In *Invited Talks Track of the 8th System Administration Conference (LISA VIII)*. USENIX, September 1994.

[4] Arnold de Leon. From Thinnet to 10baseT, from Sys Admin to Network Manager. In *Proceedings of the 9th System Administration Conference (LISA IX)*. USENIX, September 1995.

[5] Daniel J. Bernstein. *pty documentation and man pages*, 1992. Available by anonymous ftp from mojo.eng.umd.edu in /pub/misc/pty-4.0.tar.gz.

[6] Randal L. Schwartz. The chat2.pl package. Posted to comp.sources.unix, June 1991.

[7] Matt Blaze. NFS Tracing by Passive Network Monitoring. In *Proceedings of the USENIX Winter Conference*. USENIX, January 1992.

[8] David A. Curry and Jeffrey C. Mogul. *nfsstat man page*, 1993.

SQL_2_HTML: Automatic Generation of HTML Database Schemas

Jon Finke – Rensselaer Polytechnic Institute

ABSTRACT

The ongoing development of our relational database based system administration package, Simon, requires frequent reference to documentation that describes the existing database tables. To this end we have written a program that uses descriptive information stored in the database itself, to generate a WWW tree that documents each table in HTML, as well as an index page to tie the whole package together. This has made looking up table definitions simply a click or two away and has proven to be very useful. These HTML pages are now also being included in some of our program documentation of the Simon system.

Introduction

We manage many of our Unix system administration tasks via a system we have developed called Simon. The Simon system is based on a relational database, in our case, Oracle. At the start of the project, the importance of documentation of the database design was recognized so we made use of an Oracle feature that allows you to store comments on both tables and columns directly in the database. We first wrote a program that would generate a simple text file with the description and attributes of each table. As the number of tables grew, this program was modified to generate *man* (1) pages.

Along with the man pages, versions of this program were produced to generate both TeX and nroff format descriptions of the database objects for inclusion in papers and other documentation. This has served to strengthen our practice of requiring that all tables and columns be fully documented in the database before they are installed in the production system. This also prevents us from loosing the documentation, as long as the table lives, so does the documentation.

Unfortunately, with the number of tables that make up the Simon system, the man(1) based solution was not scaling well; often the key piece of information needed was the table name, and the man(1) command demanded an exact match. With the growing availability of WWW browsers, a new solution appeared. The program that generated the man pages went through yet another evolution to become **SQL_2_HTML** a program that generates HTML descriptions of each database table, and the key, an index of all the tables with hot-links to each.

HTML File Generation

For each table SQL_2_HTML is processing, we create an HTML format file. All the files are written to the current directory with the file name being the table name (with the first letter in each word of the table name capitalized), and a suffix of ".html". By following a consistent file name format, it is simple to generate references to each file from other html pages. Since all the files are in the same directory, we can use relative links to switch between tables without any worry about the actual file system path. Thus, we end up with some files like the following:

```
Disk_Acct.html Logins.html People.html
```

to hold documentation for the Disk_Acct, Logins and People tables.

Table Information

Once the file is created, we first write an HTML title and header lines, followed by the description of the table itself, as stored in the table comments part of the database. We can optionally include the table creation date and last time the table definition was changed. Another option will include the number of rows in the table, and the amount of space taken by the table. This is of course the size at the time the html page is generated (and should include the date of generation.)

It is possible to define an index on one or more columns in a table. This can provide some performance improvements by allowing the database to just traverse the index to find the desired row, rather than having to read the entire table. However, depending on the nature of the data, an index may not always help performance, and may even hurt performance in some cases. Thus, when investigating the performance of a query on a table, it is important to know what indexes exist for the table. Therefore, we consider all of the indexes on table an important part of the documentation for the table and include a list of those indexes in the HTML page.

Each index is listed by name, and with the column or columns that are indexed. As an added convenience, each column name is also a link to the specific column description, which follows later in

the HTML document. This is helpful as in some of the larger tables, the list of columns can be quite long.
This can be seen in Figure 1.[1]

One of the keys to generating source code for any text processor, including the web, is to provide proper processing for special characters. Since the people who entered the comments on tables and columns were not concerned about special characters, any comments (or other info) extracted from the database needs to be "cleaned" in order to trap the special characters. To do this, we wrote a version of `fprintf` that expands the arguments into a buffer, scans the buffer for special characters ("<", ">" and "&") and quotes them properly, and then calls `fprintf` with the "cleaned" string. Of course when we need to actually generate HTML directives, we have to call `fprintf` directly. We used a similar technique in the programs that generated LaTeX and troff documents.

Column Information

Now that we have extracted most of the general information for the table, we generate an entry for each column in the table. Along with the column name, we obtain the datatype definition for that column[2] and use these as the title (<dt>) in an HTML descriptive list (<dl>..</dl>). To assist readers in distinguishing the name from the type, we also put the column name in bold face type. We also define an anchor point at the column name, using a lower case version of the column name as a link name. This allows links to be made to the column information from within this page, or from other documents. Since we use a consistent name pattern a reference to the Username column in the Logins table would be written:

```
href="Logins.html#username"
```

The fundamental concept of a relational database is its ability to *join* two tables together using a column from each of the tables as a key. For instance, in the Logins table, there is a column called OWNER which indicates the owner of that particular Unix Login. Rather than storing the owner's name, addresses, etc, the OWNER column contains a numeric value (or key) which corresponds to the People.Id[3] column. Since many tables refer to the People.Id column and we have taken measure to ensure that all valid People.Id values can be found in the People table; the People.Id column is considered to be the **primary key**[4] of the People table. Since the Logins.Owner column refers to the People.Id column, the Logins.Owner column is considered a **foreign key** in the Logins table.

The words `Primary Key` in the heading after the data type indicate that that particular type is a primary key for that table. Alternately, if the column is considered a foreign key, this is indicated with the words `Foreign Key` followed by the name of the primary key. Since we follow a consistent pattern in file naming and the column anchor

[1]The examples shown here were displayed using Netscape Version 1.1.

[2]Number, Char(nn), Date, etc

[3]We often refer to database objects as a format like {Table}.{column} or {owner}.{table}.{column}, thus People.Id is the ID column in the People table.

[4]This is not intended to be rigorous definition of "Primary Key"

Oracle TABLE Logins

All past and present accounts, as well as reserved username and uids.

When Created: Sat Sep 10 23:06:02 1994
Last Modified: Tue May 16 17:50:23 1995
Number of Rows: 16117
Kilobytes used: 2684
Index: I_Logins$Owner Logins (owner)
Index: I_Logins$Unixuid Logins (unixuid)

username char(8)
8-character identifier, also called login name, userid, login, netname, or account; key for this table, but not necessarily unique, given different source values
unixuid number - Primary Key
Referenced by: Group_Members.Unixuid, People.Employee_Uid, People.Student_Uid,
UNIX UID; should usually be between 1000 and 32767; NULL if not UNIX
public_personal_info char(240)
Public personal information as it should appear in system list (aka gecos)
owner number - Foreign Key: People.Id
People.id of the person currently responsible for use; usually the only person who knows the password

Figure 1: Sample Table

points, we are able to make the primary key name a hot link to the actual primary key definition.

Now that we finally have the title of the column entry set, we can work on the actual descriptive entry (<dd>). If this column is a primary key, we first list all the foreign keys that refer to it. Each of these entries also act as a hot link back to their own table/column definitions. After that, we include any comments that have been saved for that column.

Relational Links

The table and column comments, and documentation generation programs have been with the Simon project from the start, however no effort was made to formally record the relationship between primary keys and foreign keys. In fact, the *SQL Language Reference Manual (version 6.0)* states:

> Currently ORACLE Version 6.0 supports the syntax of constraints, and stores all constraint definitions in the Data Dictionary. This version does not actually enforce constraint definitions (NOT NULL is currently supported.)

Until the advent of commonly available hyper-text browsers such as Netscape and Mosaic, we did not bother to record the relationships (except as text in the actual comments.)

With the start of the **SQL_2_HTML** program, these constraint definitions suddenly became relevant. Rather than trying to parse column comments for other column names[5] we can now document the **relationships** between columns and tables directly in the database. This has allowed us to not only put in links from a column to the primary key in another table, but for each primary key, provide a list of tables (with links of course) that reference it.

Fortunately, we can go back after the fact and start recording these constraints.
We can define a primary key (say for the people table) as follows:

```
Alter table People
  add (primary key ( Id ) )
```

and then set Logins.Owner as a foreign key that points to it with:

```
Alter table Logins
  add (foreign key  ( Owner )
       references People.Id)
```

Views and Sequences

Along with the tables and indexes already described, we make use of a number of *Sequences* and *Views*. As with tables, we want to document them.

A sequence is like a table with two columns defined, "currval" and "nextval''. When you select from a sequence say "Uidcount.Nextval", you will get a number that is one higher than the previous time you made that selection. This is very handy when you need a source of numbers, such as for assigning Unix Uids. In fact, sequences have a number of attributes including the step size, the direction, the maximum value, the minimum value, and if the sequence "wraps" when it hits the maximum. This can be very handy when you want to stop some processing when it hits a limit, such as running out of Unix Uids to assign.

Generation of a sequence HTML page is straightforward. We use the same file naming convention that we use for tables. Unfortunately, oracle[6] does not currently support comments for sequences. This limitation could be addressed with the creation of a table and a few views that would virtually duplicate the table/column comment functionality with only some minor syntax differences. All of the interesting information on the sequence is extracted from a table and put into the HTML file. A sequence page is shown in Figure 2.

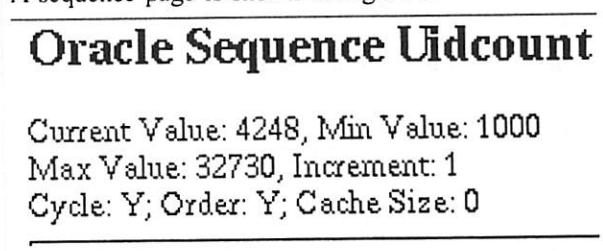


Figure 2: Sequence Page

Another very useful database object is called a view. This is a logical table based on one or more real tables. Views allow you to grant other people access to part of a table, based on any number of constraints you can define. Since views are essentially virtual tables, generation of an HTML page for a view starts out much the same way as we do for a table. The view (table) comments, the creation and last modified dates are all included, as well as the number of rows. Since views don't actually store any data (all the data lives in the real tables), there isn't any value to use for the space used.

Internally, a view is a database query of some type. While it can make use of indexes that may exist on the actual tables, you can not create an index on view, so there are no indexes to be included on the view pages. Another difference, is that you can not create constraints on the columns in a view, so the column information is limited to the column name, the data type and the column description.

[5] Yes, I was considering that option for a while. Fortunately we came up with this instead.

[6] Oracle version 6.0.37

One important thing that is missing from the view pages, is the actual definition of the view. Due to a limitation in the interface used, this information was not available for this version of the program. Once this restriction is lifted, not only will we be able to include the definition of the view on the HTML page, but we will be able to reference the underlying tables for additional comments and constraints.

Table Index

Once we have generated files containing HTML descriptions of each table, sequence and view, we then build an index page to help us get around. The index is split into three sections, one for tables, one for views and one for sequences. Each table name (or view name, or sequence name) is a link the the corresponding HTML page. The index is currently generated as a descriptive list, with the table/view name as the title (`<dt>`) information, and the table description as the list data (`<dd>`).

Even though Oracle limits the table comments to 250 characters, with the number of tables growing, the index of tables is getting to be pretty long and is becoming difficult to scroll through. We have *jump bars*[7] to get you to the start of any of the sections, but the table and view sections are simply getting to long. A short list of all tables, views and sequences without the descriptions may be useful for quick access. A look at part of the index page, as it goes from tables to views (including a jump bar) is in Figure 3.

Future Directions

There are a number of changes and extensions that I would like to do with this program. Some are basically just finishing the existing work, and others will take some more design to find the best approach.

Improve View Support

As mentioned in the section on views, I want to extract the definition of the view and include it in the page. This was not done in the initial version due to a limitation in the API[8] we use to access the database. Once we are able to extract the view definition, we want to parse the definition to identify the base tables and columns and make those as hot links to the appropriate places.

Be able to access the underlying column definitions will also allow us to extract the column descriptions to be included in the view documentation. We often did not take the time to document the column descriptions in a view since their description was unchanged from the base table.

[7]Jump bars are those horizontal lists of hotlinks in an HTML document that help you navigate within large documents. They are repeated throughout the document so you don't have to scroll too far before you can click and jump.

[8]We use a locally developed API called RSQL. It is a subroutine interface we developed to connect to the vendor application interface (OCI). In many ways it is much simpler to use than OCI, and allows us some additional networking options. On the other hand, it does not currently support the LONG datatype, which is how view definitions are stored. Since SQL_2_HTML is the first application that needs this, we may finally add the support.

Userid_Changes
A list of pending and completed requests to change RCS userids from one value to another. We assume that any pending changes have met all policy and ownership requirements.
User_Directory_Info
A table of user directory information that people may want to update for inclusion in the directory. The one row per user is enforced via an unique index
Volumes
The list of volumes that we know about and bill for. Entries are added when we detect new volumes by comparing this to afs_volumes, updated when a quota changes, and during billing.
Wtmp_Status_Log
A place to record the last time a host checked on the wtmp_status version, and the last time that version was updated.

Jump to: Tables Views Sequences

Oracle Views for Simon

Etc_Passwd
A view of the logins table, roughly equivelent to what could be found in the /etc/passwd file. It does have the addition of the SOURCE field and the OWNER field. Passwd hash is not available. Only active userids are returned.
My_Mail_Group_Name_Memberships
A view of all the mail group names that this person is in some way a member of. While alias names are normally non published, being a member of a mail group entitles you to know the name. Most columns come directly from the Mail_Group_Names table

Figure 3: Sample Index

Weak Links

In a perfect relational database, you never have more than one copy of any given data element. In practice however, I find that data is often copied between tables, especially when interfacing to some external agency. For instance, when we match the Simon student list with the Registrar's student list, we use the "student number"[9]. This is later copied to the Simon.People table, and appears in a few other places as well. No one table really "owns" the student number, yet many different table reference it. Since we can't make it a primary key, we could define a "weak link" table, to help hold these relationships together in the HTML page.

We also have relations between tables. The Simon.Students and Simon.Employees tables both feed into the Simon.People table. Since this data propagation is often of interest, being able to generate links that follow these paths would also be useful. Since the table HTML pages can be regenerated at any time, these relations would also have to be stored in the database. If this was done effectively, this could also form the basis for a data flow diagram.

Multiple Schema Support

While most of the Simon tables simply reference other Simon tables, we do have other projects that reference Simon tables, and some Simon table make references to other projects. However, due to operational requirements, we can't have all the tables from all the database users in the same directory. Not only do you have the problem of file name conflicts (although that could be solved by prepending the Owner to the table name), you have licensing restrictions to consider. While it may be fine for RPI to publicly permit the Simon schema, it would be questionable at the very least to publicly permit the schema for the Financial accounting system that we run.

One solution to this problem, would be to create a html_doc_home table that records the full path[10] or URL prefix for each Schema. In this way, when SQL_2_HTML is generating a reference to a table/column in a different schema, it can look in this table to get the appropriate prefix to make the hot link.

Database Access Status

One set of information that would be very useful when administering the Simon system, would be the table access information. This would have a list for each table and view, of who has what kinds of access. It would also have a page for each Simon user of what specific access they have.

While easy to generate, I have not yet decided if this should be included on the page with each table, or if it should be broken out into it's own tree. We need to consider the privacy and security issues involved here. Providing a potential hacker with a list of who has access to sensitive information would make it easier for them to target an attack on an individual.[11] Given the operational rather then development nature of this information, this might be better kept elsewhere (with links to the main tree of course.)

External References

Programs outside of the database also reference tables and columns. It would be nice to be looking at a particular table or column, and find out what programs and subroutines access it. Ideally you would want to be able to link to both documentation and source code[12].

On the assumption that you are going to document your programs and subroutine libraries, it is a simple matter to include links to the tables. However, I find I often want to go the other way, given a table, find out who references it. One approach to this, is create a Reference Registry table, where you can add references to the document when you are writing it, and when the table document is regenerated, these references are automatically included. Ideas on how to automate the registration would be welcome.

Including references to the source code might be a bit trickier. Currently, just about all of the Simon code[13] uses the RSQL routines to make oracle calls. These look just like `printf` statements, so it would be possible, sort of, to parse these to identify tables and columns references. Unfortunately, some of the statements use variable substitution, so you have execute the code to actually see what tables and columns are used.

```
sql("Select X,Y,Z");
sql("  from Simon.Table");
```

works, while

```
char *third_col, *tab_name;
sql("Select X,Y,%s", third_col);
sql("  from %s", tab_name);
```

requires actual execution to identify the tables and columns.

[9] A 9 digit number frequently mistaken for a social security number.

[10] At RPI, many of our systems share a common AFS file system, so a file system path is adequate for many of the references. Also, some pages can not be released to the public, so we rely on the file system access control.

[11] Security by obscurity is not a good idea, but this might keep some less obvious targets hidden.

[12] After all, isn't the source code the ultimate program documentation?

[13] At last count, around 50,000 lines of code in about 200 modules.

The idea of "scanning" the code for references does still have some appeal. A way of building a general program cross reference to trace subroutine calls would be handy. Some of this may already be handled in other software, or in packages like "ctags" for emacs. For now I may have to be content with simply documenting what I write in HTML, and making the links to the tables.

Another alternative would be to embed references in comments such as:

```
/* SQL_TABLE(Simon.Logins,Insert) */
/* SQL_TABLE(Simon.People,Delete) */
/* SQL_COLUMN(Logins.Owner,Update) */
/* SQL_COLUMN(People.Id,Select) */
```

This actually includes not only the table/column information, but some indication of what action may take place (Select, Insert, Update, Delete). This could be very useful in determining which program update or reference tables. Unfortunately, it requires adding additional comments to the programs.

Availability

As with all other parts of the Simon project, this program is available to anyone who wants it. The actual Simon HTML tables are available from a link on my home page. While we do not have a formal release, all Simon and supporting code are freely available via AFS or anonymous FTP. These are the actual working directories.

For anon FTP, connect to ftp.rpi.edu, and root around in the pub/its_release directory. In the "simon" directory, there is a README file that explains what is available and where to find it. For AFS users, take a look in /afs/rpi.edu/campus /rpi/simon.

The SQL_2_HTML program currently requires the RSQL routines. These are available for Oracle Version 6, Oracle version 7, and another site has ported them to Sybase.[14]

References

[ORACLE90] "SQL Language Reference Manual; Version 6.0", Oracle Corp, Revised February 1990..

[FINKE92] "Oracle Tools", Jon Finke, Community Workshop 92, hosted by Rensselaer Polytechnic Institute, June 13-19, 1992, Troy, NY.

Author Information

Jon Finke graduated from Rensselaer in 1983, where he had provided microcomputer support and communications programming, with a BS-ECSE. He continued as a full time staff member in the computer center. From PC communications, he moved into mainframe communications and networking, and then on to Unix support, including a stint in the Nysernet Network Information Center. A charter member of the Workstation Support Group he took over printing development and support and later inherited the Simon project, which has been his primary focus for the past 4 years. Reach him via USMail at RPI; VCC 315; 110 8th St; Troy, NY 12180-3590. Reach him electronically at finkej@rpi.edu. Find out more via http://www.rpi.edu/~finkej.

[14]At that time, Sybase did not have the table and column comment facility, although this would be trivial to implement with a few tables and views.

Decentralising Distributed Systems Administration

Christine Hogan – Synopsys, Inc.
Aoife Cox – Lockheed-Martin, Inc.
Tim Hunter – Synopsys, Inc.

ABSTRACT

Nowadays, system administration most often involves maintaining a collection of distributed, interoperating machines. The manner in which this task is carried out, however, is usually more reminiscent of a centralised computing model, with a small number of machines playing host to all of the critical system services, and constituting common failure points for the entire distributed system.

In this paper, we argue that the adoption of a distributed approach to administration of these systems is not only more natural, but can also be shown to have many practical benefits for the system administrator. In particular, we show, by example, how distributed object technology, as reflected in the CORBA (Common Object Request Broker Architecture) standard, can be used to construct a distributed administration framework, tying together services and servers on many different nodes, bringing some of the advantages of distributed systems to the systems administrator.

Introduction

As we have progressed from the age of centralised computing to that of distributed computing, so has the task of the systems administrator had to evolve from maintaining a system with perhaps one or two large servers to maintaining a large network of smaller workstations, and a number of CPU and disk servers. However the mode of operation used by system administrators themselves still tends to be that of the centralised model.

In many large sites there are one or two machines on which the rest rely absolutely for a number of services. For example, at Synopsys, a single machine is the NIS master, the mail server, an ntp time server, a boot server, the SecurID server, the console server, DNS master, NAC administrative host, and runs pcnfsd. If something goes wrong with this server it results in many and varying failures all over the network. We believe that the primary reason for the continued use of this centralised model of administration is the lack of adequate software support for decentralisation of the system administration task.

In this paper we examine the philosophies of distributed computing [1] and, in particular, distributed object technology [2] and see to what extent they could be usefully applied in the development of a systems administration toolkit. In particular, we show how the infrastructure provided by a distributed object technology, such as CORBA [3], could be used to create a framework that ties together the pieces of the existing system administrator's toolset to form a single decentralised distributed toolkit. In addition, we relate our proposal to the ongoing standards efforts, specifically the Object Management Group's proposals for the System Management CORBAfacilities [4] in the set of Core Facilities for CORBA-compliant platforms.

Background

In this section we introduce some terminology that will be used in the paper and provide some background on the research that is being performed in the field of distributed object technology. Initially we discuss each of the components, object technology and distributed systems, separately, and then examine the area that combines the two.

Object Technology

In object technology an *object* is frequently defined as a representation of a real-world entity, which has state, behaviour and identity. For example, an object could be used to represent a user. The structure of an object is generally specified using classes, where a *class* can be viewed as a template for a set of objects having a number of characteristics in common. A class will define the data that determines the state held by an object of that class, together with the operations that can be performed on that data.

There are many advantages in, and motivations for, using object technology as a systems modeling paradigm [5]. It offers a more natural way to model real-world problems through describing the behaviour of each entity in the system and the interactions between those entities.

Object-oriented design paradigms encourage modularity of code. Encapsulation of internal data

structures into objects provides the separation of interface and function from implementation, making code more maintainable and easier to enhance. Modularity and encapsulation also yield flexibility and extensibility due to the design and development of independent modules that are combined together to solve a problem. Object-oriented design allows for incremental growth of a system without major re-design, through the modularity and composibility of the components, and the ability to extend the components and the services provided by a class through the use of inheritance and polymorphism. System administration tools can also benefit from these features of object-orientation.

Distributed Systems

In this paper, when we refer to a distributed system, we mean a collection of loosely coupled processors, such as a network of workstations [1]. Some of the key advantages of distributed systems are that they provide the potential for incremental growth, load balancing, fault tolerance, high availability and reliability through replication of services. Other advantages of distributed systems include resource sharing, and new possibilities in the area of Computer Supported Co-operative Work (CSCW).

Incremental growth means that as new technology and machines become available, they can be added to the network, and obsolete machines can be removed, without any great difficulty.

With a distributed system it is also possible to distribute the load of service providing among a number of machines, spreading the load between them, and not relying on a single overloaded server. Load-balancing in this way also facilitates the replacement of servers with newer, faster machines, since only one or two services need to be rolled over to the new machine, rather than five or more. The use of a distributed system makes the maintenance of a set of distributed servers and the roll-over process easier.

Some system services, such NIS [6] and DNS [7] already implement replication for the reasons that were mentioned above. However, the replication is built into each of them separately. They neither provide nor use an infrastructure which a system administrator can conveniently utilise in order to implement replication of other services.

Distributed systems also introduce problems of their own, however, such as latency, security and the traditional lack of software. Latency is inherent in the nature of a loosely-coupled distributed system. No networking hardware today is as fast as a bus connecting two processors. Security issues include authenticating access from remote machines [8] and other people on the network eavesdropping on potentially sensitive data [9]. There is a lot of research being performed in the area of security, and some distributed systems, such as OSF's Distributed Computing Environment (DCE) [10], have very strong network security [11]. Similarly, there has been considerable research in the area of distributed operating systems over the past years, and the products that have traditionally been available do not particularly simplify the task of distributed programming. The lack of software to simplify the task of distributed programming is a well recognised problem [12]. One of the most essential services that a distributed system can supply is a location service [13]. A location service can be used by server programs to advertise themselves, and by client programs to locate the services that they require.

Recent advances in application-level software have mainly been in the area of distributed object technology, which is reaching maturity and acceptance with the release of the CORBA 2 standard and the development of the common facilities (CORBAfacilities) architecture [4]. Implementations of the CORBA standard are among the first distributed programming solutions that provide a programming interface at a high enough level to be useful.

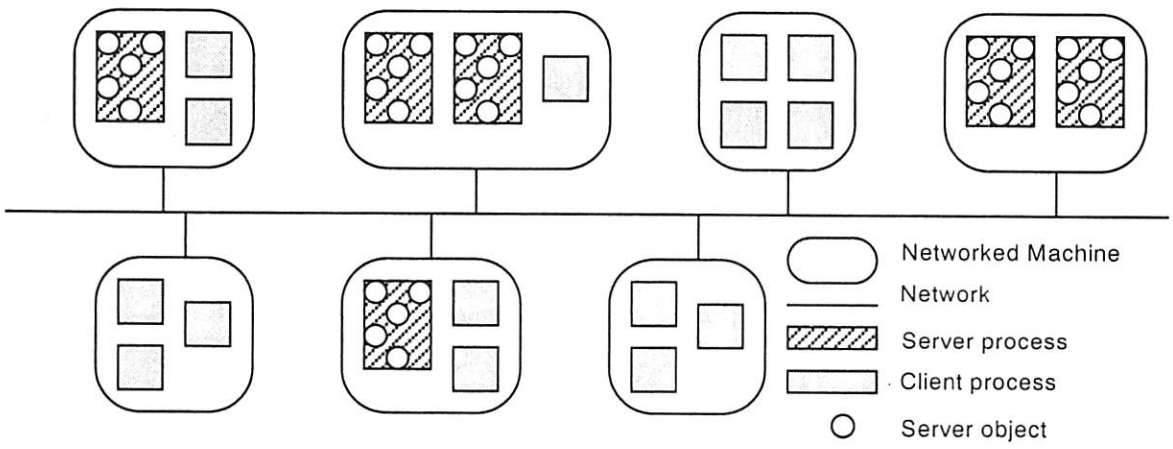


Figure 1: A distributed object support system

Distributed Object Technology

In this section we introduce the concepts behind and motivation for distributed object technology. We also mention the ways in which distributed systems and object technology have been combined in CORBA [3] and the Object-Oriented Distributed Computing Environment (OODCE) [14]. Figure 1 illustrates a distributed object-support system. Communication occurs between a client and zero or more servers, by way of object invocations. The servers with which a client communicates can reside on the same machine as the client, or a remote machine. With standard object-oriented languages, all of the objects must reside in the same address space to be able to access each other. A distributed object support system, such as those defined by the CORBA standard, provides either an interface definition language or language extensions that support the notion of distributed objects. Object can be anything from fine-grained language-level objects, to processes, to physical components of the system, such as printers. Objects should be uniformly accessible by any component of the distributed system in a transparent, location-independent manner.

In the section on distributed systems, we described how services could be distributed throughout the network using standard distributed systems technologies like DNS [7] and NIS [6]. We noted that a location service could provide a more flexible environment. Standard distributed systems architectures also lack the advantages inherent in the use of object technology, as outlined in the section "Object Technology". A distributed object support system, such as a CORBA-compliant system, combines the advantages of object technology with a distributed environment. It also provides the flexibility of a location service.

CORBA

In this section we provide a brief introduction to CORBA. Initially we describe the evolution of CORBA, and then provide an architectural overview. Finally, we briefly answer the questions of language support and availability.

Evolution

A number of different companies produced software to aid in application interworking, such as Sun's Tooltalk, Microsoft's OLE and Hewlett-Packard's SoftBench. In an effort to create a standard in high level application interworking, including across multiple platforms and network architectures, the Object Management Group (OMG) was formed, and produced the Common Object Request Broker Architecture (CORBA) specifications [3]. The CORBA specifications describe a messaging facility for a distributed object environment: a mechanism whereby each object in the environment has a standard way of invoking services of other objects in that environment.

Architectural Overview

The CORBA architecture, as specified by the OMG, is comprised of three main elements. These elements are an Object Request Broker (ORB), an Interface Definition Language (IDL), and a Dynamic Invocation Interface (DII).

The Object Request Broker (ORB) is a fundamental service that enables messaging between objects in centralised or distributed systems. The ORB handles the details of all communications between clients and servers, irrespective of the language in which they are written, or the platform on which they reside. Conceptually, the ORB handles the passing of requests from a client to a server and passing the results back.[1] The Interface Definition Language (IDL) is used to specify the interface to a given object. If a client has a handle to an object and knows the IDL interface that the object supports, the client can invoke a method on that object through the ORB. This form of invocation uses the object's static invocation interface.

The Dynamic Invocation Interface (DII) is used where the client does not know the interface to a server object in advance. The DII can be used to formulate requests at runtime. A server object will not be aware that an incoming request was performed using the dynamic interface, rather than the static one.

Given this specification a number of implementations of the CORBA standard, and in particular the ORB, are possible. For example, Orbix, from Iona Technologies, implements the ORB through three components. These components are a client library, a server library and an Orbix daemon (`orbixd`). This implementation is discussed in more detail in the section "Using CORBA". An alternative implementation might not have a daemon process, but rather implement everything in the client and server libraries.

Languages

The IDL interface definitions are compiled into a high-level language like C or C++. Other language options are available, but their availability depends on the CORBA implementation that you are using. There is not, to the authors' knowledge, an IDL to Perl compiler available, nor any compiler that will translate IDL into a high-level scripting language, such as those that are commonly used by system administrators.

However, scripts that implement the task that the system administrator wishes to incorporate into a distributed architecture, based on CORBA, can be called from the C++, or equivalent, wrappers.

[1] The handling of message passing may be implemented in separate client and server libraries, but conceptually this functionality is in the ORB.

Invoking the scripts in this manner is the way in which we envisage the CORBA architecture being utilised in the development of system administration tools.

Availability

There are a number of different implementations of the CORBA specifications commercially available. These packages include Orbix from Iona Technologies, ObjectBroker from Digital Equipment Corporation, and DOE from SunSoft. All examples and implementation-dependent details in this paper are based on Orbix.

To the authors' knowledge, there are currently no free implementations of CORBA on Unix.

Application to System Administration

In this section we address the practicalities of how we can utilise the infrastructure provided by CORBA to do the hard work of distribution. We describe how you can link existing programs or scripts into this infrastructure, with the gains being simplicity, ease of operation, flexibility, and the potential for higher reliability and availability.

Simplicity and ease of operation are due to being able to run the front-end client code on any machine, irrespective of the machine, or machines, on which the server code is run. Thus all of the tasks that are linked into the infrastructure provided by CORBA can be accomplished without having to log in to the servers themselves. This architecture also provides the flexibility to move services without having to alter the client code at all. The service is located by the CORBA location service, transparently, at run-time. Higher reliability and availability can be achieved through distribution of services over a greater number of machines, and through the potential for duplication of services.

Using CORBA

To develop a CORBA-based application, using, for example, the Orbix software described previously, requires development of client and server programs. Machines that are going to host your Orbix servers must be running an instance of the Orbix daemon. Servers are registered with the daemon. This registration communicates the presence of the server, and the command line parameters needed to launch it, to the daemon. At this point client applications can access the server by getting a handle to it through the location service.

Server Implementation

A server interface is written in IDL. Some sample IDL code is provided in Appendix 1. Skeleton C++ classes corresponding to the IDL interface(s) are then generated using the IDL compiler that is supplied with the product. The developer then supplies the method bodies for each method in the interface. This process is depicted in Figure 2.

The top-level routine of the server (i.e., `main()`) creates objects as required, and notifies the Orbix daemon that it is ready to receive incoming requests. The daemon listens for incoming requests for any of its servers and launches the appropriate server. The server executes the

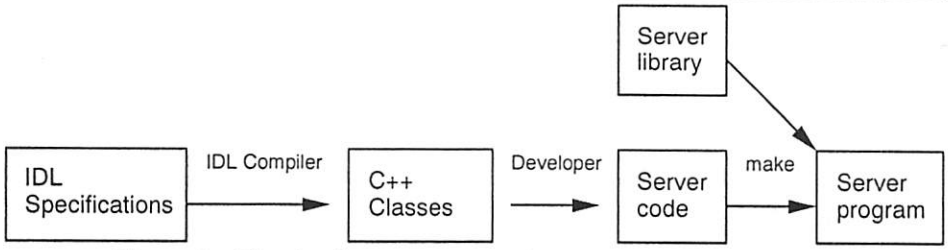


Figure 2: The development process from IDL to server program

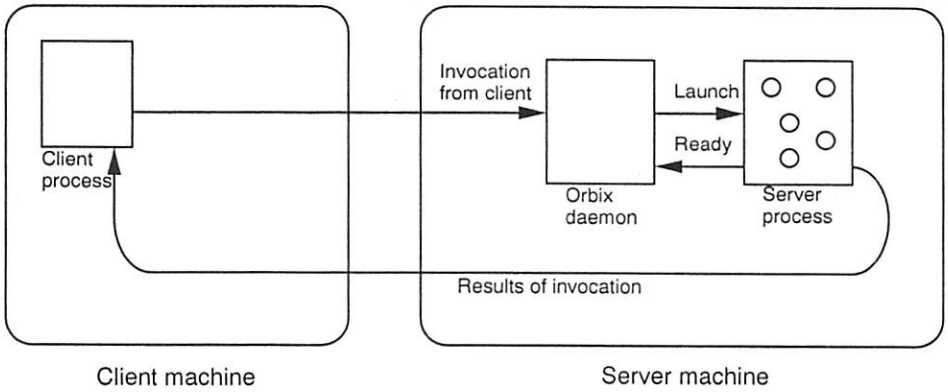


Figure 3: Launching a server

invocation request and passes the results back to the client. This process is depicted in Figure 3. The Orbix daemon acts in a similar manner to the portmapper here. Note that when a server registers itself, it registers a name for itself that the client can use when it is trying to locate an object provided by that server, and that a single server can register more than one object. Servers can also choose one of two different invocation methods. One of these invocation methods is to have a single instance of the server handle all incoming invocation requests. The other form of invocation of the server involves an instance of the server being created for each incoming method invocation, so that they are handled in parallel in different processes. This decision is made by the server when it registers itself the Orbix daemon.

The server is linked with a server library that implements the communication between the server and the daemon, including this notification of readiness. The server library also implements the communications between the server and its clients. This communication is over TCP/IP with XDR encoding.

Client Implementation

A client is linked with the client library, which handles all communication between the client and server, and between the client and the daemons. The initial communication performed by a client involves locating the server objects with which it wants to interact. Server location is performed through the Orbix `bind` operation. Some sample client code is provided in Appendix 1.

When a client calls `bind` it supplies a number of arguments to tell the ORB what kind of server object it is looking for. For example, the client can specify the name of the server process, the server object name, the interface that the object provides and the machine on which the server should reside. In other words, the client can choose to bind to a specific object associated with a named server on a particular host. If the client either doesn't know, or doesn't care about the host on which a server resides, it can specify the server name and the object interface. Equally, the client can omit the server name. If the host name is left unspecified, Orbix consults a number of its configuration files. These configuration files specify what hosts have daemons running, and the order in which to check them to find an object supporting the specified interface.

Once a client has successfully bound to a server object, it can perform method invocations on that object as if the object was in the client's own local address space. The client actually has a *proxy* [15] object in its address space, that represents the remote object, and provides the same interface as the remote object. Invocations on this object are transparently passed to the remote object in the remote server process. The library software takes care of the details of passing the invocation parameters to the server and returning the results back to the client.

Simple Operations

If the program, or script, that you want to link in to CORBA is a simple program that is supplied a series of arguments, and runs to completion, with perhaps some output along the way, then linking it into a CORBA system is simple.

The implementation involves writing a server that calls the script with the arguments that are supplied to the method call. The client program would pass the arguments that are supplied on the command line as arguments to the method call on the remote server object. In this simple case, there is little gained by using CORBA rather than `rsh`, except the ability to move the server transparently to the application, and the ability to provide backup servers that will transparently get called in the absence of the primary server.

Complex Operations

A complex operation is one that involves several servers on a number of different machines. An example of a complex operation is the creation of a new account.

In Synopsys, Human Resources generate an incoming form for each new hire that tells each of the departments the information they need to prepare for the new hire. In particular, we in Network and Computing Services (NCS), get the information necessary to create the account and to order and install equipment. The account creation is automated through a Perl script that retrieves and parses the incoming form.

At the moment, this script must be run on the overloaded server mentioned in the Introduction, for a variety of reasons, including our trust model. The script goes through a series of steps. The incoming notification, with all the details, has to be retrieved from one server. An entry for the new user needs to be created in the NIS maps on the NIS master. A home directory must be created for the user on whatever server is appropriate for their group. They will need to be added to a variety of email lists, which may involve a series of files on different machines. Also, there may be some special requirements, such as a system administrator being added into the call tracking system, or access to a database, which requires an account on another machine.

To implement a complex operation, such as the one described above, under CORBA, there is a server process on each of the machines that can be involved in the operation. Thus there would be a server process on each of the home directory servers, on the NAC administrative host, on the NIS master, on the database machine, on the machine that controls the call tracking system, on the machines that

house email lists that may need changing, and on the machine that supplies the incoming forms.

The servers should each represent one logical service. If, for example, NIS and the call tracking system reside on the same machine, they should be implemented as two separate server objects on the same machine, to facilitate moving one of the services to another machine. There may be more than one server on a single machine for a given complex operation. The client will then bind to each server object that it needs, and can invoke the operations in the appropriate order, independent of where the servers reside.

Implementing an operation like the one described above under CORBA would be advantageous in our environment, because it would obviate the need to run a large, complex script on an overloaded server. Each component of the script would be run on the relevant machine, and the script would not have to be changed if any of the services are migrated around the system. Nor would service migration necessitate a change in our trust model.

Interactive Scripts

Interactive scripts present a greater challenge, because a series of interactions between the client and the server side need to happen, with information being passed in both directions. In CORBA, communication between the client and server takes place when the client invokes a method on a server object, or the server returns the results of a method invocation to the client.

Thus to implement an interactive script within CORBA, the client needs to be more complex than a simple call, or a series of simple calls to remote services. Separate operations that return results must be identified, and implemented as remote method calls. The client contains the logic and the interaction. Thus interactive scripts require a greater redesign than non-interactive ones to be incorporated into a distributed architecture.

Caveats

In this section we discuss two caveats in the use of an infrastructure of this kind to provide system administration facilities. The first of these caveats relates to the use of persistent object support when implementing a system administration task. The other caveat relates to security issues involved with providing servers that perform system administration tasks in response to requests from the network.

Persistence

Many distributed object support platforms provide persistent object support [16] [17]. A persistent object is one that has a lifetime beyond that of the programs that access it. In these systems, application-level objects can be stored in persistent store, such as on a disk, when not in use and thus maintain their state between invocations of an application. They also survive system restarts. Persistence is not a fundamental component of distributed object support, nor of CORBA, but it can be a useful feature for many applications. It may also superficially appear to be useful for system administration applications.

For example, the system could keep a persistent database of what account(s), if any, each person had on each machine in the distributed system, along with all other electronic resources that the individual used, including mailing lists, call tracking systems and database access. This information would be stored in the persistent object associated with that individual. When the "remove user" method was invoked on that persistent object, all the information would be immediately available, which would make it simple to write the script to perform the deletion.

However, we believe that it would be a mistake to use persistence to store system state. Consider what happens when the state of the system is modified either manually, or by a program or script that does not update the persistent state of the objects that represent the altered system state. Worse still is a scenario in which the persistent state of an object gets corrupted, but is still used to determine the behaviour of the machine in some way.

We came to the conclusion that it must be possible to fix the state of the persistent store so that it reflects the state of the standard system files. It should also be possible to do this without rebooting the machine, since the introduction of a new technology should not detrimentally affect the availability of the system as a whole. Thus it must be possible to re-initialise the persistent state of the distributed object system at any time from the system files. Given the need for that feature, and the possibility for conflicting updates due to the state of the files being changed without a corresponding change in the persistent state, we came to the conclusion that persistence was not useful for this application. We also felt that it introduced extra points of failure into the system, and thus was not only not useful, it would be a mistake to employ persistence.

Security

The standard mode of operation of Orbix, and, we believe, the other implementations of CORBA, is not especially secure. If the Orbix daemon is not run as root, all the servers are clearly launched with the same user ID as the Orbix daemon. In this case, the system administration utilities cannot operate. If the Orbix daemon is running as root, the daemon tries to launch a server with the user ID of the remote user, if that user exists on the system on which the server is to be run. The user ID of the remote user is passed with the invocation request by the client library. The security implications of the

server naively believing information that comes in, unauthenticated, off the network are obvious.

There are hooks for applying filters, including an authentication filter, to servers on a case by case basis. This authentication filter could require some form of strong authentication before allowing a server to be invoked. However, the overhead of implementing such authentication may be sufficiently large that it outweighs any advantages of implementing the service over CORBA. Further work is being performed in the area of security, including the provision of standard security services within the framework of the CORBAfacilities, but it remains to be seen what these will provide.

Related Work

The Object Request Broker, as defined by the OMG, forms a part of an overall Object Management Architecture (OMA), which specifies a model for constructing distributed object applications. The ORB is the key communications element of the architecture. The architecture, in addition, defined CORBAservices (formerly known as Common Object Services) and CORBAfacilities [4] (formerly called Common Facilities).

CORBAservices specify standard interfaces to basic functions commonly required in building (distributed) applications. These basic services include object naming, event notification and transactions. CORBAfacilities specify standard interfaces to functions that are required for building applications both in specific domains and across domains. CORBAfacilities include a proposal for a system management facility. The CORBAfacilities for system management will comprise a set of IDL interfaces, and hence a standardised collection of servers providing system management functionality. Guidelines for possible facilities have been outlined by the OMG. However, the actual interfaces for most facilities, including those proposed for system management have not yet been specified.

Our work does not advocate any standard set of facilities or interfaces. In this paper we merely outlined how we believe the infrastructure provided by CORBA can be utilised as framework for providing distribution for existing administrative scripts and tools.

Conclusions

The infrastructure provided by a CORBA-compliant systems is potentially useful for building a decentralised system administration toolset. However, in the absence of an IDL to Perl[2] compiler, it is unlikely to become a tool that is regularly employed by system administrators. Equally, we would expect the system administration community to have reservations about using it, unless the security issues are resolved, or the administrators individually take the view that it's not that much worse than running NIS.

However, we do believe that the architecture has potential, and that it may be something that is worth watching for in the future. In particular the proposed CORBA security services offer interesting potential for deploying a standard security system across all pertinent applications in an environment. Implementations of these security services, when available, will offer convenient access to the building blocks of security for applications developers and system administrators alike. We believe that the availability of these tools will help promote electronic security.

[2]Or other scripting language.

Acknowledgments

Paul E. provided encouragement, advice and practical help from day one, right through to the end, and for that, and for his tolerance we thank him. Paul A. also deserves a special mention for advice, and a push in the right direction at a crucial time, along with much needed encouragement. Our many proof-readers, and Jeff in particular, were of enormous help in straightening the paper out - our thanks to Beth, Arnold, Ted, Dave and Laura for their help.

Author Information

Christine Hogan is the security officer at Synopsys, Inc., in Mountain View, California. She holds a B.A. in Mathematics and an M.Sc. in Computer Science, in the area of Distributed Systems, from Trinity College Dublin, Ireland. She has worked as a system administrator for six years, primarily in Ireland and Italy. She can be reached via electronic mail as chogan@maths.tcd.ie .

Aoife Cox is a research scientist at the Lockheed Martin Artificial Intelligence Center in Palo Alto, California, where she leads the object management infrastructure team on Simulation Based Design (SBD) - a major ARPA project aimed at supporting distributed concurrent engineering. She holds B.A. and M.Sc. degrees in Computer Science from Trinity College Dublin, Ireland, where she spent a number of years working with the Distributed Systems Group in the Computer Science Department. Her research interests include distributed computing, software reusability and concurrent engineering. She can be reached via electronic mail as acox@maths.tcd.ie .

Tim Hunter is a systems administrator at Synopsys, Inc. His current focus is on remote systems administration. He previously worked as a sysadmin for, and received his degree in Electrical and Computer Engineering from, the University of Colorado at Boulder. He can be reached via electronic mail at tim@synopsys.com .

[1] G.F. Coulouris and J. Dollimore. *Distributed Systems Concepts and Design.* Addison-Wesley, 1988.

[2] Rodger Lea and James Weightman. Supporting Object-Oriented Languages in a Distributed Environment: The COOL Approach. In *Proceedings of the Technology of Object Oriented Languages and Systems Conference*, July 1991.

[3] OMG. Common Object Request Broker Architecture. Technical Report OMG Document 93.12.43, rev 1.2, Object Management Group, Inc., December 1993.

[4] OMG. Common Facilities Architecture. Technical Report OMG Document 95.1.2, rev 4.0, Object Management Group, Inc., January 1995.

[5] Bertrand Meyer. *Object Oriented Software Construction.* Prentice Hall, 1991.

[6] Hal Stern. *Managing NFS and NIS.* Nutshell. O'Reilly and Associates, Inc., 1991.

[7] Paul Albitz and Cricket Liu. *DNS and BIND.* Nutshell. O'Reilly and Associates, Inc., 1992.

[8] Aviel D. Rubin. Independent One-Time Passwords. In *Proceedings of the 5th UNIX Security Symposium.* USENIX, June 1995.

[9] Matt Blaze and Steven M. Bellovin. Session-Layer Encryption. In *Proceedings of the 5th UNIX Security Symposium.* USENIX, June 1995.

[10] Open Software Foundation. *Introduction to DCE*, 1991. Part of licensed DCE documentation.

[11] Rich Salz. Dce. Bay LISA, April 1995. This is the 2nd edition of his LISA VIII talk.

[12] Andrew S. Tanenbaum. *Modern Operating Systems.* Prentice Hall, 1992.

[13] Aoife Cox. An Exploration of the Application of Software Reuse Techniques to the Location of Services in a Distributed Computing Environment. Master's thesis, Distributed Systems Group, Dept. of Computer Science, University of Dublin, Trinity College, September 1994.

[14] John Dilley. OODCE: A C++ Framework for the OSF Distributed Computing Environment. Technical report, Hewlett-Packard Laboratories, 1994.

[15] Marc Shapiro. Structure and Encapsulation in Distributed Systems: the Proxy Principle. In *Proceedings of the 6th International Conference on Distributed Computing Systems*, May 1986.

[16] Vinny Cahill, Seán Baker, Gradimir Starovic, and Chris Horn. Generic Runtime Support for Distributed Persistent Programming. In *OOPSLA (Object-Oriented Programming Systems, Languages and Applications) '93 Conference Proceedings*, 1993.

[17] Roy H. Campbell and Peter W. Madany. Considerations of Persistence and Security in Choices, an Object-Oriented Operating System. In *Proceedings of the International Workshop on Computer Architectures to Support Security and Persistence of Information*, May 1990.

Appendix: IDL Definitions and Client Code

```
---- user_accounts.idl ----
//
// This file describes the classes
// associated with user accounts
//
module User_Accounts {
    //
    // If an exception is raised we use
    // this class to return the reason
    //
    exception reject {
        // The exception data
        String reason;
    };
    //
    // This is the interface to the home
    // directory class, with its visible
    // attributes
    //
    interface Home {
        attribute String path;
        attribute long uid;
        attribute long gid;
        // The method - create home dir
        void initialise() raises(reject);
    };
    //
    // Described the interface to the
    // User class - operations and
    // visible attributes
    //
    interface User {
        attribute long uid;
        attribute long gid;
        attribute String username;
        attribute String shell;
        attribute String gecos;
        attribute Home home;

        void initialise() raises(reject);
        void remove() raises(reject);
        void disable() raises(reject);
        void reactivate() raises(reject);
    };
    //
    // Interface to the SystemUser class.
    // It's a specialised case of the User
    // class.  Only the SystemUser object
    // itself can modify the real_user
```

```
    // attribute - clients can't
    //
    interface SystemUser : User {
        // Inherits the interface from
        // User and adds one read-only
        // attribute to that interface.
        readonly attribute User
                            real_user;
    };
    //
    // Interface to the UserDatabase class
    // An instance represents the passwd
    // file.  Operations are called on
    // an instance of this class by User
    // objects to get the passwd file
    // modified.  Some User objects may
    // call it twice - e.g. SystemUser
    // objects, to disable privileged
    // and unprivileged accounts, for
    // example.
    //
    interface UserDbase {
        readonly attribute String
          passwd_file;
        attribute String
          default_encrypted_passwd;
        // Re-read the passwd file
        void reinitialise()
          raises(reject);
        // Disable an account
        void disable(in String username)
          raises(reject);
        // Reactivate disabled account
        void reactivate(in String
          username) raises(reject);
        // Delete a user entirely
        void remove(in String username)
          raises(reject);
        // Add a new user
        void add(in User user)
          raises(reject);
    };
};
```

```
---- client.cc ----

// This program runs on the client and
// requests the server to perform
// operations on a UserDatabase object.

#include <user_accounts.hh>

main()
{
    User_Accounts::UserDbase *user_db;
//
// This is where the bind magic happens
// Bind to a server object that has the
// UserDbase class interface from the
// User_Accounts module, that lives in a
// server called ''user_database_server''
// on any machine.
//
    user_db =
        User_Accounts::UserDbase::_bind(
        ''':user_database_server''');
//
// Standard C++ exception-handling
// syntax.  TRY something, and CATCH the
// exceptions.
//
    TRY {
        ....
    }
    NONE {
        ....
    }
    CATCH(UserAccounts::reject,
      rej_except) {
        cout << ''reason: ''
             << rej_except->reason
             << ''==>[ignored: n]<==''';
    }
    CATCHANY {
        ....
    }
    ENDTRY

    ....
}
```

SPM: System for Password Management

Michael A. Cooper – University of Southern California

ABSTRACT

Most UNIX operating systems are not delivered with adequate facilities to address password management in medium to large, heterogeneous environments. The System for Password Management (SPM, pronounced *spam*) provides multiple levels of user verification beyond the normal password verification procedure, both local and centralized password database management, a consistent command interface across multiple platforms and multiple password database types, fast and efficient updates to large NIS *passwd* databases, proactive password checking, and password aging. SPM is intended to replace the **passwd**, **yppasswd**, **nispasswd**, **chfn**, **chsh**, **ypchfn**, **ypchsh**, and **rpc.yppasswdd** programs, although it can run without replacing these programs.

Introduction

The ever increasing number of UNIX users and rapidly expanding popularity of direct Internet access is quickly increasing the number of novice computer users. Subsequently, there is also an ever expanding number of people trying to subvert computer security systems. The large number of novice computer users provides more opportunities than ever for intruders to illegally access computer systems as well as greatly increasing the demand on system support staffs.

The tools provided by most UNIX vendors usually are not well designed to handle this situation in large, heterogeneous environments. Most UNIX systems provide inconsistent command interfaces, in some cases none at all, to change a user's password, full name (GECOS) information, or login shell. Sun's Solaris 2.4 operating system, for example, requires each user to explicitly know where their password information is stored and run the appropriate command (i.e., **passwd** for /etc/passwd, **ypppasswd** for NIS, and **nispasswd** for NIS+) Furthermore, Solaris 2.4 does not even provide a facility for user's to change their NIS or NIS+ full name (GECOS) field or login shell.

Most UNIX systems also do not provide any form of proactive password checking. This can lead to users using very weak passwords which are vulnerable to being compromised by software such as *crack*.

When user's identities are verified for purposes of changing their password, most systems only require the user to enter their current password. If an intruder has compromised an account's password, then the usual methods for maintaining password security, such as password aging, will fail to deny access to the intruder. The ability to query the user for multiple types of information known only to the user and the password system (such as the user's SSN, mother's maiden name, etc.) normally will prevent intruders from maintaining access to compromised accounts.

There are also some serious deficiencies in how changes to the NIS passwd database are performed. One of the most severe is that only one change to the NIS *passwd* file can be made at a time. While the NIS *passwd* file is being changed, all other update attempts are refused. On a master NIS server with a large number of passwd entries (e.g., more than 1000), it can take a small, but substantial amount of time to re-write the passwd file. If a large number of users, such as a class of new users, attempts to change their passwords at about the same time, most users will be unable to because of this single-step update mechanism.

SPM was designed to address these deficiencies. It currently supports */etc* style files and NIS/YP on SunOS 4.1.x (Solaris 1.x), SunOS 5.x (Solaris 2.x), and AIX 3.2.x. The code is very portable to new UNIX operating systems and support for additional password facilities can be added in a very straightforward manner.

The USC Environment

In order to provide more insight into the the design and implementation of SPM, it is useful to describe the environment at the University of Southern California (USC) that spawned SPM.

There are approximately 8,000 nodes attached to USC networks. Of that number, about 1,300 are UNIX machines, 2,000 are Macintosh machines, and 3,800 are PC class machines.

Computing at USC is separated into two main types – Academic Computing (student computing, research computing, basic computing, etc.) and Administrative Computing (financial records, student records, etc.). University Computing Services (UCS) is charged with providing support for Academic

Computing directly to researchers and students, providing support for computing resources owned by departments, and support of the campus networks and Internet connections.

UCS supports a number of specific computing facilities:

- The Research Computing Facility (RCF) consists of Sun, SGI, Convex, and IBM compute servers and is dedicated to providing a high-end environment for performing research.
- The Student Computing Facility (SCF) consists of about 10 Sun servers, and several hundred Sun workstations for use by graduate and undergraduate students for class related work.
- The Basic Computing Facility (BCF) consists of a single Sun server dedicated to providing free access to email, USENET News, and WWW to any USC faculty, staff, or student.

Most hosts supported by UCS are configured to use NIS[1] (formerly YP). While most hosts are split into small, independent NIS domains, there is one large NIS domain that that currently has over 20,000 *passwd* entries and over 1,700 *group* entries. (Most of the users in this domain are part of our Student Computing Facility.) Needless to say, this has allowed us to experience some of the major flaws in both the NIS design and implementation discussed in this paper.

Vendor Provided Interfaces

Before describing SPM in detail, it is useful to be familiar with the general functionality provided by most vendor's to allow user's to change their password information. Since it is not the intent of this paper to provide an in-depth analysis of such interfaces, most attention will be on generalities with a few examples.

Description of /etc files

The most basic UNIX password database provided by virtually all UNIX vendors is the **/etc/passwd** file. This file normally contains seven fields, each separated by a semi-colon:

- **username** A user's login name.
- **password** A user's encrypted password.
- **uid** A numeric user identification number.
- **gid** A numeric identification number specifying the user's primary group.
- **fullname** A text field contain the user's full name and/or other descriptive information. This field is sometimes referred to as the *GECOS* field.
- **home** The user's home directory.
- **shell** The user's login shell.

On most systems, there is a **passwd** command that a user can run to change the **password**, **fullname**, and **shell** fields of their password entry. For a user to change their **password** in **/etc/passwd** they would normally run

```
% passwd
```

The **passwd** program will prompt the user for their current password and confirm that it matches what's currently set in **/etc/passwd**. The user is then prompted for a new password and asked to re-enter the new password to confirm that it matches what the user typed the first time. Most **passwd** implementations perform a few basic evaluations of the new password to test for acceptability. Usually this only involves checking to insure the password is of a minimum length.

Once a new password is confirmed with the user, the **passwd** program will then lock the **/etc/passwd** file, usually by creating a **/etc/ptmp** lock file, then it will update **/etc/passwd** itself. Usually this is done by writing out a new version of **/etc/passwd** into the **/etc/ptmp** lock file, then installing **/etc/ptmp** as **/etc/passwd**.

On some newer systems there exists an auxiliary password database file, which is often called a *shadow* password file and is usually named **/etc/shadow**, that contains encrypted passwords as well as password aging information. The *shadow* password file is normally readable only by "root". This prevents normal users from accessing all the encrypted password fields and running a program such as **crack** to guess user passwords. On such systems, the **password** field in **/etc/passwd** will only contain something like "x" to indicate the encrypted password is really located in **/etc/shadow**.

Description of NIS/YP

The Network Information Service (NIS, formerly called Yellow Pages) developed by Sun Microsystems and licensed to most major UNIX vendors, provides distributed access to to various system databases, such as **automount** maps, **group**, **passwd**, **hosts**, and others. The NIS protocols are based on a client/server ONC RPC architecture.

Hosts utilizing NIS are grouped into NIS domains. Each NIS domain has a single NIS master server and may optionally have a number of NIS slave servers. The NIS master contains the source of all databases. The source databases are ASCII files that are normally identical to their /etc equivalents and, in fact, are often maintained and built directly from their standard /etc locations. For instance, on most systems the **/etc/passwd** file is actually the default location of the NIS **passwd** database source file on the NIS master.

NIS databases, or *maps* as they are usually referred to, are stored in *ndbm*(3) files. Changes to NIS maps are made by modifying the ASCII source file for a particular map using an editor such as *vi*(1) or by using a third-party program of some type or in

[1] We hope to move to NIS+ as soon as our internally developed user account management system is updated.

the case of the **passwd** map, through the **rpc.yppasswdd** server. Once the source file is updated, the *ndbm*(3) files are updated by running *make*(1) in the **/var/yp** directory. The *make*(1) rebuilds the affected *ndbm*(3) databases from scratch – there is no partial update mechanism provided in NIS. Once the *ndbm*(3) databases are updated, the *ypserv*(8) process on the NIS master (the local machine where the database was just updated) is notified of the update. Any NIS slave servers that are configured are notified that a map has been updated. When the *ypserv*(8) process on each NIS slave server is notified of an update, they consult with the NIS master server to check to see if the master has a newer version of the map, and if so, transfers the entire map to local disk storage.

The **passwd** database in NIS has a special set of protocols and servers who provide a mechanism to update NIS **passwd** data. When a NIS user wants to change their password, they run **yppasswd**[2] The user is prompted in a fashion very similar to that described in the previous section regarding normal **/etc/passwd** configurations. However, instead of reading and writing password information from **/etc/passwd** directly, the **yppasswd** command exchanges data with the **rpc.yppasswdd** server via ONC RPC on the NIS master host. The **rpc.yppasswdd** reads and writes the NIS **passwd** source file, which is usually either **/var/yp/src/passwd** or **/etc/passwd.** Changes to the NIS **passwd** file are done by creating a lock file (such as **/var/yp/src/ptmp** or **/etc/ptmp**), then creating a new **passwd** file with the requested changes, installing the new **passwd** file, and then removing the lock file. Once the **passwd** file is updated, **rpc.yppasswdd** will by default run a *make*(1) to update the **passwd** NIS map as previously described. For systems with medium to large numbers of users, it is a common practice to disable this feature and to run the NIS *make*(1) via a **cron**8 job once per hour or so.

Most NIS systems are configured by default such that when one user changes their **passwd** information, nobody else can change their own password information until the **passwd** map has been updated and pushed out to all NIS slave servers. This can take anywhere from a few seconds to several hours depending on the size and configuration of a given NIS domain. It is possible to modify the behavior of the standard **rpc.yppasswdd** and how the NIS Makefile **/var/yp/Makefile** operates such that users are only blocked from **passwd** changes while the actual NIS **passwd** source file is being updated. However, this can take from a few seconds to several minutes depending on how many users are in a given NIS domain. If several dozen users try to change their passwords at once (which often happens to user's who are taking a class on UNIX for the first time), most will be prevented from doing so due to the single threaded behavior of the NIS **passwd** database.

[2]Some systems provide a **passwd** program that has the ability to handle NIS password changes.

Description of NIS+

The Network Information Service Plus (NIS+) was created by Sun Microsystems as a replacement for NIS. NIS+ was designed to address many of the problems found in NIS. It specifically is intended to support larger, more diverse configurations as well as providing a much higher level of security and a much improved administration interface. It can be configured to support older NIS clients, but at the cost of being only as secure as NIS allows.

Like NIS, NIS+ has a client/server architecture utilizing ONC RPC as its communications protocol. Hosts configured for NIS+ are grouped into NIS+ domains. Unlike NIS, each NIS+ domain can be part of an tree structured enterprise namespace similar to the Domain Name System (DNS).

Each NIS+ domain has a primary server and may optionally have any number of secondary servers. Each NIS+ server runs the *nisd*(1m) program which provides service to NIS+ clients. Updates to NIS+ databases can be accomplished through a command line interface, a GUI interface, and through the NIS+ C API. Updates can be made to any server, either the primary or any secondary. Changes are automatically propagated to all NIS+ servers for a given domain. Unlike NIS, updates are made by propagating the changed information and not an entire database.

NIS+ databases are stored on disk in binary format and then loaded into memory by *nisd*(1m) at startup. Locking is done on an individual record basis and not on a whole database basis. This means that NIS+ servers can accept multiple changes to the **passwd** database simultaneously.

Description of SPM

SPM was designed to address the deficiencies in operation and functionality of password management found in most UNIX implementations. It is intended to provide a consistent interface to password management to users and system administrators on an enterprise-wide scale. It normally is used to replace vendor supplied programs such as **passwd, yppasswd, nispasswd, chsh, ypchsh, chfn, ypchfn,** and **rpc.yppasswdd**. SPM is a client/server based system that utilizes ONC RPC for communication.

Consistent User Interface

The SPM user interface presents a consistent look and feel across multiple versions of UNIX and multiple types of password facilities such as **/etc** files and **NIS.** This allows users to not have to focus on the details of how to change their password information on different UNIX systems. A nice

benefit of this is that users tend to spend a little more energy on what they are doing (choosing a secure password) instead of how they are doing it.

Enhancements for Large NIS Domains

In an environment where there are a large number of users in a single NIS domain, updates to the NIS **passwd** database can take a significant amount of time. The issue of users being locked out of making changes whenever any one user makes a change is also a major problem. To address these issues, SPM supports the ability to take password changes and quickly store them as transactions which are later processed asynchronously. This allows multiple users to change their password information simultaneously.

Security Features

There are a number of features of SPM that address different security issues.

User Verification

SPM supports the ability to query the user for multiple types of verification information beyond simply asking for a user's current password. This allows system administrators to have more confidence in the identity of a user whenever a user is forced to change their password.

SPM allows the system administrator to specify via the **VerifyList** variable in **spm.conf** which, if any, additional types of verification information are required. The system administrator can also specify that all or only a certain number of the verification information be verified correctly with the user.

The information used for verification is located in *Personal Data* databases on either or both the local host where the user runs the **spm** command and on a remote host specified by the system administrator. See the section on **Databases** for a description of the contents of the *Personal Data* database.

Password History Checking

SPM supports the ability to keep a history of each user's passwords. Each time a user changes their password, the *Password History* database is checked to see if the new password has been used before by that user. If it has been, the password is rejected.

When a successful password has been chosen, it is stored in its encrypted form in the *Password History* database. The system administrator specifies how many passwords per user are kept in the history database.

Password Checking

In addition to password history checking, SPM provides an extensive set of password checking procedures. When a user changes their password, the password is subjected to a series of rules to test how vulnerable it is to common algorithms used to guess passwords. The rules are partially based on most realistic checks performed by the **crack** software. The rules include checks against the user's username, full name field, a large dictionary, minimum length, character type mix, and other assorted rules. Passwords that fail any of these tests are rejected and the user is prompted to try another password.

Password Aging

Password aging[3] is the ability to specify various parameters about passwords on a per user basis. The parameters control such things as how often a user can or must change their password.

Security of SPM

A number of decisions were made during the initial design of SPM that impact the overall security of the SPM system itself. The basic thinking was that the design be as secure as possible, but not so secure that the time to implementation was severely compromised. What resulted is a system that is no less secure than the standard NIS configuration, but not as secure as we would like. We felt we did not have the time, or the will to implement a fully secure system that utilized some form of full encryption or public key access control.

The work being done to implement standard forms of encryption at the packet and RPC level also lead us to believe that our time is better spent concentrating on other aspects of security and functionality. One such possibility is Sun's *Secure RPC* [1] which allows the use of DES or Kerberos authentication and encryption in ONC RPC based applications. Unfortunately, *Secure RPC* is not yet widely available so SPM does not support it. A future version of SPM may incorporate *Secure RPC* depending on how available it becomes in the future.

The standard NIS configuration utilizing the ONC RPC based **rpc.yppasswdd** server is what we considered the weakest form of security in standard password management systems available today. When using NIS, the client (**yppasswd**) and the server (**rpc.yppasswdd**) exchange requests using un-encrypted ONC RPC calls. The client program sends the user's current (old) password to the server in un-encrypted, clear-text format in an RPC call. Changes to a user's password entry are sent by sending a **struct passwd** structure. This means that when changing a user's password, the old password is sent clear text and the new password is sent in its encrypted format. Anybody watching the network packets for such a session would see the user's old password in plain text and the user's new password in encrypted form. This would allow them to gather encrypted passwords on which they could use password guessing software such as **crack.**

[3]While SPM supports a *Password Aging* database, password aging support itself is not yet fully integrated into SPM.

In SPM, a similar approach to NIS is taken. The user's old password is sent plain text to the server (**spmd**) for verification. This is necessary in order to verify the user's password history and also prevents a rogue client from guessing passwords by querying **spmd.**

When changing password information in SPM, only the information being changed is sent. That is, if a user is changing their *login shell* then only the new *login shell* data, along with some verification data, is sent to the server. In NIS, the entire **passwd** structure is sent. This can allow a packet sniffer to obtain more data about a user than is possible in SPM.

In SPM, special care is given to the handling of *Personal Data* information. When a user changes their password, the **spm** program sends a query to the **spmd** program on the *Personal Data* server host. The server (**spmd**) responds by returning a code indicating no records for the specified user exist, or returns a *Personal Data* (**struct PDinfo**) structure that has a single "x" in every field for which information exists. The client then queries the user for each bit of information available and then sends the user-supplied information to the server for verification. While this information does travel plain-text to the server, this process does not allow a rogue client to download data from the *Personal Data* database.

The **spmd** program also limits access to itself to specific hosts and networks. The variables **SPMaccess** and **PDaccess** in the **spm.conf** file control this access. The **PDaccess** variable controls what hosts and networks can make *Personal Data* queries. The **SPMaccess** variable controls what hosts and networks can query all the SPM databases except for the *Personal Data* database.

Databases

SPM maintains and utilizes a number of databases. All databases are stored in *ndbm*(3) format in subdirectories under **/var/spm**. Database files are named by domain name and are located in subdirectories A domain name is defined by whatever naming facility is in use on a SPM client. Usually this is your **NIS** or **NIS+** domain name. On hosts that don't run a naming service, the default domain name is **local**.

An example configuration would be a site with a *sales* domain and an *eng* domain, the data for the two will be stored separately for each type of database. In this case, there would be the following database files:

```
/var/yp/age/sales.{dir,pag}
/var/yp/age/eng.{dir,pag}
/var/yp/person/sales.{dir,pag}
/var/yp/person/eng.{dir,pag}
/var/yp/pwdhist/sales.{dir,pag}
/var/yp/pwdhist/eng.{dir,pag}
```

This feature permits a single **spmd** server to support multiple domains.

Database Administration

The **spmadm** command is the primary tool available to system administrators to administer the SPM databases. This tool provides the ability to directly read and update all SPM databases. It can be used to lookup and delete individual records, as well as retrieve and store records in "raw" format. An administrator can use **spmadm** with the **-set** *key=value* option to add or modify a specific record. The **-write** option can be used to initially load a SPM database, such as the *Personal Data* database.

Personal Data

The *Personal Data* database contains information used by SPM to verify a user's identity. The type of data intended for this database are bits of knowledge, that when used together with each other, provide a higher level of confidence that a user is who they claim to be.

The following fields are currently supported:

- **SSN** The user's Social Security Number or other type of identification.
- **DOB** The user's Date Of Birth.
- **MomName** The maiden name of the user's mother.
- **HomeZIP** The user's Home ZIP code.

Each record is keyed by a username.

The information is stored as text fields in *ndbm*(3) databases usually located under **/var/spm/person**. Sites may choose to store whatever ASCII data they wish in any of these fields. However, there is no support at this time to customize the prompts presented to users when asked for each bit of information. Modifying the source code is the only means by which to do this, though this can be done fairly easily.

Password History

The *Password History* database contains a record of passwords that users have used before. The information is stored as text fields in *ndbm*(3) databases usually located under **/var/spm/pwdhist**.

Each password is stored in its original encrypted form. Each record is keyed by username. The data field for each record contains a colon separated list of previously used passwords in most-recently-used to least-recently-used order.

The system administrator specifies, through a configuration file, the maximum number, 10 by default, of passwords to maintain in each entry. When this limit is reached, the oldest password for a given entry is deleted.

Password Aging

The *Password Aging* database contains records that allow sites to specify certain policies regarding password changes.

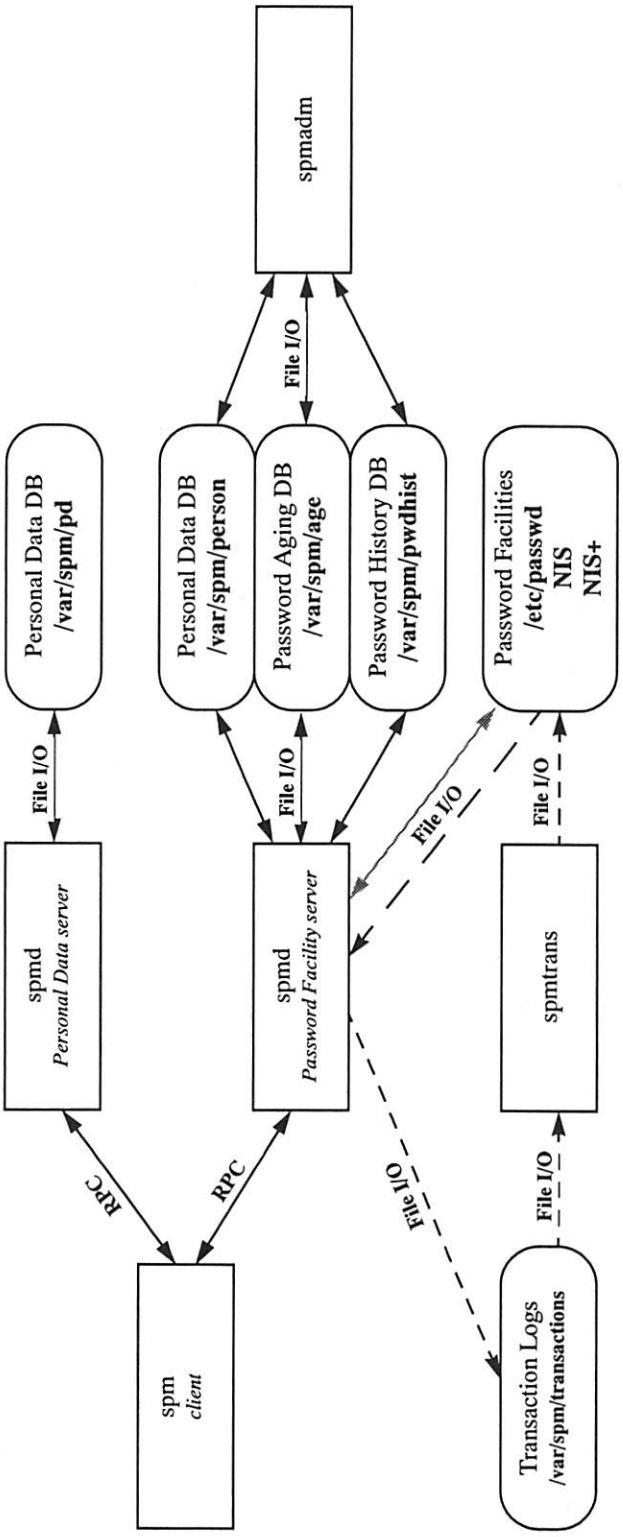


Figure 1: SPM Component Overview

The following records are currently supported:

- **ModTime** Last time a password was modified.
- **MinChg** The minimum number of days before a user can change their password again.
- **MaxValid** The maximum number of days a password is valid for before a user must change their password.
- **WarnTime** The number of days a user is warned to change their password before some type of action is taken once **MaxValid** days is reached.
- **InActive** The number of days an account can be unused before a user must change their password.
- **ExpireDate** The absolute date that the account expires.

Architecture

The SPM software consists of a client and a server which use ONC RPC as a communications protocol and several stand-alone programs that use direct file I/O. Figure 1 provides an overview on how the various parts of SPM interact.

The **spmd** program is an RPC-based server that processes requests from the **spm** client program. It processes requests to access multiple password facility databases (e.g. **/etc/passwd** and **NIS**) as well as requests for the *Password Aging*, *Password History*, and *Personal Data* (verification) databases.

The **spm** program is the interface between end users and the **spmd** server. It allows users to change their password, full name (GECOS) field, and login shell. Normally the **passwd**, **yppasswd**, **nispasswd**, **chfn**, **ypchfn**, **chsh**, and **ypchsh** programs are replaced by symbolic links that point to **spm**.

The **spmtrans** program is used to process transactions created by **spmd** if the system administrator has enabled transaction logging. It uses direct file I/O to access transactions, each of which is stored in a separate file on the system where **spmd** and **spmtrans** are run. Usually **spmtrans** is run periodically (such as once per hour) via the *cron*(8) facility on an NIS master.

The **spmadm** program is used to administer the *Personal Data*, *Password Aging*, and *Password History* databases. Its purpose is to provide a system administrator a means by which data in the various SPM database can be loaded, viewed, modified, and deleted. It uses direct *ndbm*(3) file I/O on a specified SPM database.

Configuration

The **spm**, **spmadm**, **spmd**, and **spmtrans** programs all support the ability to configure SPM variables via the command line and via the **spm.conf** configuration file. Command line options override any value specified in **spm.conf**.

All SPM programs utilize a single **spm.conf** configuration file on each host. This file allows the system administrator to configure such things as SPM database locations, servers, and access controls, password facility files, transaction logging flags, and verification information flags.

On startup, each program searches for this file in a compile-time specified set of locations. Usually the locations are as follows:

```
/var/local/conf/spm.conf
/usr/lsd/conf/spm.conf
```

The first file found is parsed. See the *spm.conf*(5) man page in **Appendix A** for further details.

The **spm** program will do one of three things – change a password, a full name, or a login shell – depending on how it is invoked. The program's behavior is specified either by a command line option or by the name of the program it was invoked as. **Table 1** shows the matrix of what triggers each type of behavior.

Table 1: Behavior of **spm**

Command Line	Program Name	Behavior
–chpw	**passwd** **nispasswd** **yppasswd**	Change Password
–chfn	**chfn** **ypchfn**	Change Full Name
–chsh	**chsh** **ypchsh**	Change Login Shell

User Examples

The **spm** program is the primary end-user interface to SPM. It can be used to change a user's password, full name (GECOS), and login shell. It is designed to act like most UNIX **passwd**, **chsh**, and **chfn** commands whenever possible.

In the following example, a user named **smith** is changing his password. The system that **smith** is logged into is configured to run NIS/YP. The **spmd** server on the NIS/YP master is configured to do transaction logging and the SPM *personal data* server has **smith's** social security number on file. Here's what the session would look like:

```
alcor.usc.edu(1): passwd
Changing nis password for smith on yp1.usc.edu . . .
Current Password:Go4Sleep
Please enter your Social Security Number: 123456789
New Password:$iAmHome
Retype new Password:$iAmHome
Your request to change your password information has
been queued for later processing.
This change may take up to two hours to take effect.
alcor.usc.edu(1):
```

The message about being "queued for later processing" is triggered by **spmd** on **yp1.usc.edu** being configured to do transaction logging. The message that "This change may take up to two hours to take effect" is triggered by **spmd** on **yp1.usc.edu** also being a NIS master and configured to not push out a new **passwd** map itself.

In our next example, our test user **smith** is changing his office phone number and the comments field of his full name information on the same host once again:

```
alcor.usc.edu(3): chfn
Changing nis finger (GECOS) for smith on yp1.usc.edu . . .
Current Password:$iAmHome

Default values are printed inside of '[]'.
To accept the default, press the <RETURN> key.
To specify a blank entry, type the word 'none'.

Full name (e.g. 'John W. Smith') [John Smith]: RETURN
Office Location (e.g. 'SAL 125') [UCC 209]: RETURN
Office Phone Number (e.g. '740-2957') [02957]: 5551234
Home Phone Number (e.g. '213-777-3456') [none]: RETURN
Comments (e.g. 'My student account') [Student]: PhD Student

        Full Name: John Smith
   Office Location: UCC 209
Office Phone Number: 5551234
  Home Phone Number:
         Comments: PhD Student

Is this information correct [Yes]? RETURN
Your request to change your finger (GECOS) information
has been queued for later processing.
This change may take up to two hours to take effect.
alcor.usc.edu(4):
```

In our final example, our restless user **smith** is now going to change his login shell from **/usr/usc/bin/tcsh** to **/usr/bin/csh:**

```
alcor.usc.edu(7): chsh
Changing nis login shell for mtest on yp1.usc.edu . . .
Current Password:$iAmHome

Here is a list of shells you may choose from:
        /bin/sh
        /bin/csh
        /usr/bin/sh
        /usr/bin/csh
        /usr/usc/bin/bash
        /usr/usc/bin/tcsh
        /usr/usc/etc/admshells/newpwd
        /usr/usc/etc/admshells/ftp-only
        /usr/usc/etc/pwd/forcepwd
        /usr/usc/etc/ftp-only
        /bin/ksh
        /bin/rksh

Old Shell: /usr/usc/bin/tcsh
New shell: /usr/bin/csh

You have selected "/usr/bin/csh" as your new shell.
Is this information correct [Yes]? RETURN
Your request to change your login shell information
has been queued for later processing.
This change may take up to two hours to take effect.
alcor.usc.edu(8):
```

Operation

The **spmd** program is invoked by *inetd*(8) the first time a request for SPM service is received. Before each RPC request is processed, a check is performed to see if the client is authorized to make such a request. Access is controlled by use of the **SPMaccess** and **PDaccess** variables as described in the *spm.conf*(5) man page and in the **Security of SPM** section.

The **spm** command is invoked by a user directly (on the command line) or indirectly (another program such as *login*(1m) runs it). Upon startup, the program first determines what information should be changed as described in the **Configuration** section. Next **spm** determines which *Password Facility* (i.e **/etc files, NIS, NISplus**) to use by searching all supported *Password Facilities*[4] and using the first facility the user's *username* appears in. A command line option also allows a user to override this behavior and specify which specific *Password Facility* they wish to use.

The search is done by calling *Password Facility* specific lookup routines. For instance, the NIS function that is used is **NIScheck**(). It performs a *yp_match*(3N) library call to lookup a given user in NIS.

The **spm** program next determines what *domain name* to use in future requests. This name is also obtained in a facility specific manner. In the case of NIS, the NIS *domain name* is used. In the case of **/etc files** being the facility, the predefined domain name **local** is used.

Next, the **spm** program determines the name of the host that is the *SPMserver* (sometimes referred to as the *Facility Server*) for the *Password Facility* that has been selected. This host is where **spm** expects to contact the **spmd** server. This is also done through use of a facility specific routine. In the case of NIS, the *yp_master*(3N) library function is used and returns the name of the NIS master host for the local hosts' NIS domain. In the case of the **/etc** facility, the name of the local host is returned.

The **spm** program next will setup an RPC connection to the **spmd** program on the *SPMserver*. The *SPMserver* is then pinged by sending an RPC **NULLPROC** request to **spmd** to determine if the server is available. If the ping fails, an error is displayed and the program exits. This is done so that a user is not prompted for lots of information only to be told the server is "unavailable".

The user is next prompted to enter their **Current Password**. Once the user does so, the password is sent off to the *SPMserver* for verification. The **spmd** program on the *SPMserver*

[4]Which facilities are supported and the order the facilities are searched is determined at compile-time.

checks the password against the user's current password in the *Password Facility* specified in the request by **spm**. If that fails, the password is then checked against the password of "root" (on the *SPMserver*). If the password was successfully matched against the password for "root", a special flag is set. The results of these comparisons are then returned to the **spm** client. If the results were successful **spm** continues on, otherwise an error is displayed and the program exits.

If the user is changing their *full name* or *login shell* entries, they are next prompted for the new information as shown in the **User Examples** section. Default values are taken from their current entries in the *Password Facility* being changed.

If the user is changing their *password* and the user did not supply the "root" password previously, the **spm** program will try to verify the user's identity using the information found in the *Personal Data* database on the *Personal Data* server (PDserver). The *PDserver* is a host running **spmd** that is either specified by the system administrator in the **spm.conf** file or else is the same as the *SPMserver*. A query is first sent to the *PDserver* to determine what types of *Personal Data* are available for the specified user. The user is then prompted to enter each type of information the *PDserver* indicated it had on file. Each type of information is checked, as it is supplied by the user, by sending it to the *PDserver* for verification. If the information is incorrect, the user is told so and prompted to try again. When the information is correctly supplied, **spm** prompts the user for the next type of information and the process continues. The user is not allowed to proceed until all verification information is correctly supplied. The user will be asked a maximum number of times for a given type of information. If they cannot correctly supply the information within that maximum, an error is returned and the program exits.

Once a user successfully completes the PD verification process they are prompted to enter a **New Password**. The **New Password** is then subjected to a series of tests that are described in the **Password Checking** section. The new password is then sent to the *SPMserver* to be checked in the *Password History* database to see if the user has used this password previously. Upon successful completion of this stage, the user is then prompted to **Retype New Password**. This password is compared against the first one the user supplied to insure that they both match.

Once the new *password*, *full name* or *login shell* information is successfully entered, the **spm** program sends a change request with the new information to the *SPMserver*. The *SPMserver* then applies the changes immediately or the changes are logged for later application if **Transaction Logging** is enabled.

If the user has changed their *password*, the *SPMserver* also adds the user's old *password* to the *Password History* database and also updates, or creates if needed, the user's *Password Aging* database entry.

If the facility being changed is **NIS** and the system administrator has specified a value for the **NISPostCmd** variable in the **spm.conf** file (usually the value is **"cd /var/yp && make passwd"**), then that command is then run to push out the changes to all NIS servers.

Transaction Logging

The *Transaction Logging* feature in SPM allows updates to the NIS *passwd* database to be quickly accepted by **spmd** and batched up for later processing. This allows multiple users to simultaneously submit *passwd* change requests without being denied service because the NIS *passwd* database is locked by another user as is the case in a standard NIS environment.

The sysadmin enables *Transaction Logging* via the **TransFacList** variable in the **spm.conf** file. This variable contains a list of *Password Facilities* which should use *Transaction Logging*.

Transactions are created by the **spmd** server and later processed by the **spmtrans** program which is usually run once per hour via the *cron*(8) facility. If *Transaction Logging* is enabled when **spmd** receives a change request, the request is stored in its own file in a *Password Facility* specific directory under **/var/spm/transactions**, i.e., the directory used for the NIS facility is **/var/spm/transactions/nis**.

The transaction file name is in the format:

Format.ID.Revision

The *Format* portion of the name indicates what format the contents of the file are in. Currently, only the **spmtr1** format is used. The *ID.Revision* parts form a unique and ascending number such that new transactions have a greater numeric *ID.Revision* value than older transactions. This is done so that **spmtrans** knows the order in which to process the transactions. The *ID* portion is created using the value returned by the *time*(2) function. The *Revision* portion is a counter that starts at "0" and is used to help generate a unique file name.

Once the filename is generated, a call to *open*(2) with the **O_CREAT** (create file) and **O_EXCL** (exclusive file creation) flags specified. If the *open*(2) fails due to an **EEXISTS** (file exists) error, another call to *time*(2) is made, the *Revision* number is incremented, and another *open*(2) call is made. This process repeats itself until the exclusive file creation succeeds or a maximum threshold value (specified at compile-time) is reached.

Once the transaction file is successfully created, the file is locked and the transaction data is written in the following form:

UserName:Type:OldValue:NewValue

The *UserName* field indicates which *username* in the password database the transaction should be applied to. The *Type* field indicates what type of information the transaction applies to. Currently, the valid names are **pwd**, **shell**, and **gecos**. The *OldValue* field is used to indicate what the contents of the database was when the transaction was created. If this field does not match what is in the database when the transaction is processed, the transaction is discarded. The *NewValue* field contains the information that should be placed into the database.

Once **spmd** writes the data to the transaction file, the file is unlocked and closed. A message is logged via the *syslog* (3) facility detailing the change and a another message is returned to the client (**spm**) stating that the request "has been queued for later processing".

The transaction files are processed when **spmtrans** is run (usually via the *cron* (8) facility). Upon startup, **spmtrans** searches for and reads a **spm.conf** configuration file. The **TransFacList** variable is used to determine which *Password Facilities* should be checked for transactions that need to be processed. A directory search for transaction files (as described previously) is performed for each *Password Facility* transaction directory (**/var/spm /transactions/***facility*). Each transaction file that is found is locked and then read into memory. The transactions are sorted in memory by *username*, in newest to oldest order, which is determined by the numeric value of the transaction's filename.

Once all transaction files are read, all transactions for each type of possible change operation (**PWD, GECOS, SHELL**) are processed in one database update operation, if possible. e.g., The **/var/yp/src/passwd** file would be written once with all **PWD** (password) changes, then it would be written again with all **GECOS** changes (if any).

During the processing of transactions, the current value found in the facility database is compared against the *OldValue* field in the transaction. If the two fields do not match, the transaction is considered out-of-date and discarded. Since the transactions are processed in newest to oldest order, this eliminates having to do multiple updates to the same entry and insures only the most recent change is applied.

Once the transactions are all processed, the transaction files are unlocked and each transaction that was successfully applied is deleted.

Performance

Probably the most important aspect of performance for a password management system is how quickly a user can change their password. While there is not much empirical data available, some rough comparisons are still possible. During peek usage at USC, a class of 30 users in a single NIS domain with 20,133 users running **spmd**[5] with *Transaction Logging* enabled can change their passwords simultaneously with only about a 15 second wait once the change request is sent to **spmd**. The same requests take another 15 seconds or so to be applied to the NIS *passwd* source file by **spmtrans**. In a standard NIS environment with this same scenario, only one user at a time would be able to change their password.

Related Work

SPM is not the first attempt to supplement or replace vendor password systems. A number of public domain programs has been available for several years. Some comparisons to some of these programs are made in the following sections.

Comparison to npasswd

The **npasswd** [2] program has been distributed since about 1989. Like SPM, **npasswd** is intended to replace a vendor supplied **passwd** program. It supports SunOS 4.x **/etc/passwd** and NIS/YP password facilities.

The main feature this program supports is the ability to check a user's new password to "disallow simple-minded passwords" in a manner similar to SPM's pro-active password checking. It also allows a system administrator to use a configuration file to specify different policies and parameters that affect its password checking routines.

Unlike SPM, however, **npasswd** does not support the ability to change a user's *full name (GECOS)* or *login shell* fields. There is also no facility for handling password history or additional user verification procedures.

Comparison to passwd+

The **passwd+** [3] program has been in existence in various forms since at least 1992. Like **npasswd** it supports extensive pro-active password checking as its main feature. It also has an even more extensive configuration file that allows a system administrator to finely tune password checking policy and local environment characteristics. It does include support for changing *full name* and *login shell* information. However, it appears to only support **/etc/passwd** facilities, though a more current version of the software may support NIS. It also does not support SPM features such as password history, password aging, and additional user verification.

Current Status

SPM has been in campus-wide use at USC since August, 1994. It currently supports SunOS

[5]The server running **spmd** in this case is a heavily loaded Sun SPARCserver 490 with 128MB of memory running SunOS 4.1.3.

4.1.x, SunOS 5.x (Solaris 2.x), and AIX 3.2.5 operating systems. The */etc* file password files and NIS/YP are the currently supported *Password Facilities*. Support for HP/UX and IRIX are planned in the near future. There is some support for NIS+ in place, but that code has not been tested yet.

Future Directions

The issue of RPC encryption remains an important item awaiting a good solution. It is hoped that something such as Sun's *Secure RPC* software or DCE Security Services becomes widely available.

The verification database should be abstracted to allow sites to specify arbitrary types of verification information and the labels presented to users for that information. For example, not every site has SSN data. Some sites might have "employee numbers" they want to use instead. A system administrator should be able to configure SPM to prompt for **Employee Number:** instead of for a **Social Security Number:**.

It might also be useful to add the ability for the system administrator to specify different types of options for the pro-active password testing. This would require a bit of an overhaul of the password checking routines, but should not be a major problem.

Conclusions

Since SPM's initial deployment at USC, it has eliminated major problems with inconsistent user interfaces, bugs in vendor provided programs, enhanced security, and allowed us to continue to grow and better support a large NIS environment. Its portable design and implementation allow us to support multiple platforms and password facilities both now and into the future.

Author Information

Michael Cooper has been working on UNIX systems for 12 years. He has worked in the Research, Development, and Systems Group of University Computing Services at the University of Southern California since 1985 and has also been an independent UNIX consultant since 1988. Reach him via U.S. Mail at: University Computing Services University of Southern California Los Angeles, California, 90089-0251. Reach him electronically at mcooper@usc.edu .

Availability

SPM will be available for FTP from usc.edu:/pub/spm by September 1, 1995.

References

1. "Secure RPC," *Solaris 2.4 System Administrator Answerbook*, pp. 46-48.
2. Clyde Hoover, *README for npasswd 1.6*, March 1990.
3. Matt Bishop, *README for passwd+*, June 2, 1992.

NAME

spm.conf – Password Management System configuration file

SYNOPSIS

set *variable* = *value*
include *filename*

DESCRIPTION

The **spm.conf** file is read by the **spm, spmadm, spmd,** and **spmtrans,** programs. Lines beginning with "#" are ignored.

KEYWORDS

The following is the list of valid keywords and their syntax:

set *variable=value*

Set a variable named *variable* to have a value of *value. variable* may be one of the following:

AgeDir The name of the Age database directory. (Default is **/var/spm/age**)

AskTries
The maximum number of times a user is asked for information. (Default is **5**)

LocalPostCmd
The command to run after updating a local facility database. (Default varies)

LocalPwdFile
The name of the local facility password file. (Default is usually **/etc/passwd**)

LocalPwdLock
The name of the local facility password lock file. (Default is usually **/etc/ptmp**)

LocalShadowFile
The name of the local facility password shadow file (on supported systems). (Default is **/etc/shadow**)

LockRetryInt
The intervel (in seconds) to retry obtaining a lock on a piece of data. (Default is **10**)

MaxPWHent
The maximum number of old passwords that are kept. (Default is **20**)

NISPostCmd
The command to run after a successful update of any NIS data. (Default is none).

NISPwdFile
The name of the NIS password file. (Default is **/var/yp/src/passwd**)

NISPwdLock
The name of the NIS password lock file. (Default is **/var/yp/ptmp**)

NISdir The name of the top level NIS directory where NIS data is stored. (Default is **/var/yp**)

PDaccess
Access list for personal data. (See the **ACCESS INFORMATION** section) (Default is none)

PDserver
The hostname of the personal data server. (Default is the name of the facility server).

SPMaccess
Access list for general SPM queries (See the **ACCESS INFORMATION** section) (Default is none)

SPMserver
The name of the SPM server to send queries to. (Default varies according to the facility being used.)

PdDir The name of the personal data database directory. (Default is **/var/spm/pd**)

PwdHistDir
The name of the password history database directory. (Default is **/var/spm/pwh**)

TransDir
The name of the directory where SPM transaction files are stored. (Default is **/var/spm/transactions**)

TransFacList
The comma seperated list of facilities to enable transaction processing on. e.g. **local, nis** (Default is not to do transaction processing.)

VerifyList
List of verification types to check. (See the **VERIFICATION section)** (Default is **SSN, DOB, MomName, HomeZIP**)

VerifyNum
The number of verification items that the user must successfully supply. (See the **VERIFICATION section)** (Default is **2).**

include *filename*
Read the SPM configuration file *filename* and parse like any other **spm.conf** file.

ACCESS INFORMATION

The **PDaccess** and **SPMaccess** variables control access to **spmd.** The variables have the form:

type=value1,type=value2,...

type can be either **host** or **net.** If *type* is **host** then *value* should be the name of a host. If *type* is **net** then *value* should be either the name of a network or the IP address of the network.

An example is:

host=karls.usc.edu,net=engnet

VERIFICATION

The **VerifyList** and **VerifyNum** variables control most of the verification proceedures in **spm.** If **VerifyNum** is greater than 0, then **spm** will require the user to supply additional information when changing their password. The **spm** program will query the personal data server to see what, if any, information is available for the user. The user is then prompted to supply what information the personal data server said was available. If no information is available, then no additional information is asked for from the user. The **VerifyNum** variable controls the number of verify information types that must be answered correctly for the verification to be successful.

The **VerifyList** variable specifies which types of verification information are required and the order in which to prompt the user for such information. It is a comma seperated list of information types:

TYPE	DESCRIPTION
SSN	Social Security Number
DOB	Date Of Birth
MomName	Mother's Maiden Name
HomeZIP	Home ZIP code

FILES

/var/local/conf/spm.conf /usr/lsd/conf/spm.conf – SPM config files

SEE ALSO

spm(1), spmadm(8), spmd(8), spmtrans(8)

AUTHOR

Michael A. Cooper,
University Computing Services,
University of Southern California.

BUGS

NAME

spm – System for Password Management user interface

SYNOPSIS

spm [**-chpw|-chsh|-chfn**] [**-facility** *facname*] [*username*]

passwd [*options*]

chsh [*options*] [*newshell*]

chfn [*options*]

DESCRIPTION

spm changes a user's password, login shell, or full name (GECOS) information. If **spm** is run as **passwd** or with the **–chpw** option, the user's password is changed. If run as **chsh** or with the **–chsh** option, the user's shell is changed. A *newshell* value may be specified on the command line. If run as **chfn** or with the **–chfn** option, the user's full name (GECOS) is changed. In all cases, the user must always specify the current password for *username* or the password for **root** (on the **spmd** server host) for verification purposes.

If the user's password is being changed, the user may be prompted to supply additional information for verification purposes. What additional information is asked for is based on the local configuration of the SPM system and the information available for *username.*

New passwords should be six to eight characters long, consisting of both lower and upper case letters. It's further recommended that at least one non-alphanumeric character (i.e. **!@$%^&*()-+**) be included. **spm** will reject passwords that fail to pass a series of basic tests or previously used passwords.

If no *username* is specified, then the username of the user running **spm** is used.

Normally **spm** will determine what password facility (i.e **files, NIS, NISplus)** *username* first appears in and will change the appropriate value in that name service. The **–facility** option may be used to override this behavior.

On startup **spm** searches for configuration files to read. The first file found is used and the rest discarded. The config list is:

/var/local/conf/spm.conf
/usr/lsd/conf/spm.conf

After parsing the configuration file **spm** will then prompt the user for the current password to *username.* **spm** then connects to the **spmd** server responsible for the selected password facility. In the case of

local /etc files, the localhost is contacted. For NIS and NIS+, the master server for the local domain is contacted. The facility server is asked to verify the current user's password. If verification information is enabled in the **spm.conf** file, then the personal data server (by default the facility server) is then contacted and queried to find out what information is available for *username*. The user is then prompted to supply the information which is then sent to the personal data server for verification. The user is then prompted to supply the new information for what is being changed (i.e. password, full name, or shell). The new information is then sent to the facility server for updating. The user is then told of the success or failure of the update.

OPTIONS

–chfn Change full name (GECOS) information.

–chsh Change login shell.

–chpw Change password.

–facility *facname*
Change information in password facility *facname*.

FILES

/etc/passwd – Local password file
/etc/shadow – Local file containing passwords
/var/local/conf/spm.conf /usr/lsd/conf/spm.conf – SPM config files

SEE ALSO

spm.conf(5), spmadm(8), spmd(8), spmtrans(8)

AUTHOR

Michael A. Cooper,
University Computing Services,
University of Southern California.

DIAGNOSTICS

BUGS

NAME

spmadm – System for Password Management administrative command

SYNOPSIS

spmadm [*options*] **–type** *datatype* **–delete** *key*

spmadm [*options*] **–type** *datatype* **–set** *key=value*

spmadm [*options*] **–type** *datatype* **–user** *username* **–show**

spmadm [*options*] **–type** *datatype* **–read**

spmadm [*options*] **–type** *datatype* **–write**

options are:

[**–agedir** *dirname*] [**–debug**] [**–domain** *name*] [**–file** *filename*] [**–keys**] [**–pddir** *dirname*] [**–pwdhistdir** *dirname*] [**–type age|pd|pwh**] [**–user** *username*]

DESCRIPTION

spmadm performs administrative operations on the System for Password Management (SPM) databases on the local host. The program must have read and/or write access to the SPM which usually means it

must be run as "root".

The **–delete** *key* argument will delete the data in the *datatype* database with a key of *key*.

The **–set** *key=value* argument will add/modify the data with key *key* in the *datatype* database to have a value of *value*. The *key=value* argument is specific to the database type. See the **OPTIONS** section for details.

The **–show** argument will display the information for *username* from the *datatype* database.

The **–read** and **–write** arguments will read (write) the raw database named *datatype* from standard input (output) or from *filename* if *–file* is specified. Each output (input) line is in a format specific to *datatype*. The first field is usually used as the key to the database entry. The **–write** argument should only be used in emergencies. The **–set** argument should normally be used to add/modify database entries. See the **OPTIONS** section below for more details.

OPTIONS

–agedir *dirname*
Use *dirname* as the name of the AGE database directory.

–debug Enable debugging messages.

–delete *key*
Delete database entry with key *key*.

–domain *domname*
Set the domain name to *domname*. The default is the local hosts' domainname.

–file *filename*
Specify name of file to read/write. Default is standard input/output.

–keys Show database key word field.

–pddir *dirname*
Use *dirname* as the name of the Personal Data database directory.

–pwdhistdir *dirname*
Use *dirname* as the name of the password history database directory.

–read Read and display all raw database entries.

–show Show database entry.

–set *key=value*
Set database entry with key *key* to have a value of *value*. If a database entry with *key* does not exist, a new entry is created. If an entry already exists, then it is updated. The *key* and *value* are database specific.

The format of *key* and *value* for the **pwh** database is:

user=user:pw1:pw2:pw3:...

Each *pw* field should be an encrypted password string.

The following applies to the **pd** database:

SSN=*NNNNNNNNN*
Social Security Number is set to *NNNNNNNNN*.

DOB=*MM-DD-YY*
Date Of Birth. Format is Month-DayOfMonth-Year in numeric values. (e.g. **4-28-94**)

MomName=*Name*

Set Mother's Maiden name to *Name.*

HomeZIP=*code*
Set Home ZIP code to *code. (e.g.* **90210**)

The following applies to the **age** database:

MinChg=*days*
Set the minimum number of days allowed between changing passwords to be *days.*

MaxValid=*days*
Set the maximum number of days a password is valid for to *days.*

WarnTime=*days*
Set the number of days of warning a user is given before their password expires to be *days.*

InActive=*days*
Set the number of days an account may be unused before being disabled to be *days.*

ExpireDate=*date*
Set the absolute date that the user's password expires to be *date.* The *date* is of form **MM-DD-YY [HH:MM[:SS]]**

–type *datatype*
Use database named *datatype.*

–user *username*
Specify username named *username.*

–write Write raw data to a database.

FILES

/var/local/conf/spm.conf /usr/lsd/conf/spm.conf – SPM config files

SEE ALSO

spm(1), spm.conf(5), spmd(8)

AUTHOR

Michael A. Cooper,
University Computing Services,
University of Southern California.

DIAGNOSTICS

BUGS

NAME

spmd – System for Password Management server daemon

SYNOPSIS

spmd [**–|+bg**] [**–agedir** *dirname*] [**–debug**] [**–localpostcmd** *command*] [**–localpwdfile** *filename*] [**–localpwdlock** *filename*] [**–nispostcmd** *command*] [**–nispwdfile** *filename*] [**–nispwdlock** *filename*] [**–pddir** *dirname*] [**–pwdhistdir** *dirname*] [**–transdir** *dirname*] [**–transfaclist** *facility1,facility2,...*]

DESCRIPTION

spmd is the server for the System for Password Mangement. It performs actions based on queries from the **spm** command. These actions include checking a user's current password, listing what verification information is available for a user, checking verification information for a user, and changing a user's password, full name (GECOS), and login shell.

On startup **spmd** searches for configuration files to read. The first file found is used and the rest discarded. The config list is:

/var/local/conf/spm.conf
/usr/lsd/conf/spm.conf

(See the **spm.conf(5)** manual for the syntax of **spm.conf** files.) After parsing the configuration file any arguments supplied on the command line are parsed which will override what was specified in the **spm.conf** file.

When a client sends a query to check a user's password or update their password, full name, or login shell, the query includes what password facility to use. This is normally /etc files, YP/NIS, or NIS+. **spmd** will then perform the requested operation on the appropriate files for that password facility. The databases for age, personal data, and password history are all **ndbm(3)** databases named **/var/spm/age/***domain*, **/var/spm/pd/***domain*, and **/var/spm/pwh/***domain* respectively. The *domain* name is specified by the client sending the query and is usually the client's NIS or NIS+ domainname or **local** if no domainname is configured on the client.

If transaction processing is enabled (see **spm.conf(5)**) for the facility specified in a request, then any request to change a user's password, full name (GECOS), or shell is logged to a transaction file for later processing. Transaction files are placed in **/var/spm/transactions.** See **spmtrans(8)** for more information.

OPTIONS

–agedir *dirname*
Use *dirname* as the name of the AGE database directory.

–bg|+bg
Disable (**–bg**) or enable (**+bg**) running certain tasks such as post commands (that are normally run after updating things like YP/NIS databases) in the background. The default is to background if **–debug** is not specified.

–debug Enable debugging messages.

–localpostcmd *command*
Run the command *command* after every update to the **local** password facility.

–localpwdfile *filename*
Use *filename* as the password file for the **local** password facility.

–localpwdlock *filename*
Use *filename* as the lock file for the **local** password facility.

–nispostcmd *command*
Run the command *command* after every update to the **NIS** password facility.

–nispwdfile *filename*
Use *filename* as the password file for the **NIS** password facility.

–nispwdlock *filename*
Use *filename* as the lock file for the **NIS** password facility.

–pddir *dirname*
Use *dirname* as the name of the Personal Data database directory.

–pwdhistdir *dirname*
Use *dirname* as the name of the password history database directory.

–transdir *dirname*
Use *dirname* as the name of the directory to use for transactions.

–transfaclist *facility1,facility2,...*
Enable transaction processing on *facility1,facility2,...*

FILES

/etc/spmd.pid – PID file

/var/spm/age – Directory of age data
/var/spm/pd – Directory of personal data data
/var/spm/pwh – Directory of password history data
/var/spm/transactions – Transaction directory
/var/local/conf/spm.conf /usr/lsd/conf/spm.conf – SPM config files

SEE ALSO

spm(1), ndbm(3), spm.conf(5), spmadm(8), spmtrans(8)

AUTHOR

Michael A. Cooper,
University Computing Services,
University of Southern California.

DIAGNOSTICS

BUGS

NAME

spmtrans – System for Password Management Transaction Processor

SYNOPSIS

spmtrans [**–debug**] [**–|+verbose**] [**–logstdout**] [**–transdir** *dirname*] [**–transfaclist** *facility1,facility2,...*]

DESCRIPTION

spmtrans is the transaction processing program for the System for Password Management (SPM). The purpose of **spmtrans** is to process the transactions created by **spmd(8).** The **spmd(8)** server creates transactions in password facility specific directories under **/var/spm/transactions.** Each transaction is stored as a seperate file in order to minimize locking and rollover issues.

On startup **spmtrans** searches for configuration files to read. The first file found is used and the rest discarded. The config list is:

/var/local/conf/spm.conf
/usr/lsd/conf/spm.conf

(See the **spm.conf(5)** manual for the syntax of **spm.conf** files.) After parsing the configuration file any arguments supplied on the command line are parsed which will override what was specified in the **spm.conf** file.

The **TransFacList** variable (from **spm.conf** or via the command line **-transfaclist** variable) is used to determine which password facilities should be checked for transactions that need to be processed.

For each facility with transaction logging enabled, a scan is performed of a directory name of the form:

TransDir/FacilityName

For the **NIS** facility, the directory name might be:

/var/spm/transactions/nis

Every transaction file found in the facility directory being scanned is locked and then read into memory. The transactions are organized in memory by username, in newest to oldest order. Once all transaction files are read, all transactions for each type of possible change operation (**PWD, GECOS, SHELL**) are performed in one database update operation, if possible. e.g. The **/var/yp/src/passwd** file would be written once with all **PWD** (password) changes, then it would be written again with all **GECOS** changes (if any). Transaction files are unlocked and deleted after each successful database update.

Each transaction file name is of the format:

format.ID.Revision

The *format* portion of the name indicates what format the contents of the file are in. Currently, only the **spmtr1** format is supported. The *ID.Revision* parts form a unique and ascending number. New transactions have a higher numeric *ID.Revision* value, then older transactions. This is used to determine the order of newest to oldest transactions.

The content format of a **spmtr1** transaction file is of the format:

UserName:*Type*:*OldValue*:*NewValue*

The *UserName* field indicates which username in the password database the transaction is for. The *Type* field indicates what type of information the transaction applies to. Currently, the valid names are **pwd, shell, gecos** . The *OldValue* field is used to indicate what the contents of the database was when the transaction was made. If this field does not match what is in the database when the transaction is processed, the transaction is discarded. The *NewValue* field contains the information to place into the database.

OPTIONS

–debug Enable debugging messages.

–logstdout
Send logging (messages) to stdout. The default is to log to the **syslog(8)** facility.

–transdir *dirname*
Use *dirname* as the name of the directory to use for transactions.

–transfaclist *facility1,facility2,...*
Enable transaction processing on *facility1,facility2,...*

–verbose|+verbose
Disable (**–verbose**) or enable (**+verbose**) verbose messages. The default is to enable verbose messages.

FILES

/var/spm/transactions – Directory of transactions
/var/local/conf/spm.conf /usr/lsd/conf/spm.conf – SPM config files

SEE ALSO

spm(1), spm.conf(5), spmadm(8), spmtrans(8)

AUTHOR

Michael A. Cooper,
University Computing Services,

University of Southern California.

DIAGNOSTICS

BUGS

AGUS: An Automatic Multi-Platform Account Generation System

Paul Riddle – Hughes STX Corporation
Paul Danckaert – University of Maryland, Baltimore County
Matt Metaferia – MCI Telecommunications

ABSTRACT

As a computer network grows to accommodate thousands of users across many different types of hardware, account generation becomes an increasingly large burden on system administrators. Computer vendors often provide tools to create accounts; however, these tools tend to be specific to the vendor's hardware and OS, and most of them do not provide the degree of automation necessary to avoid a lot of repetitive work.

In a large network with a constantly changing user base, an automated account generation system would greatly ease the burden on the system administrators. Many such tools exist, but none of them provide support for many varied platforms such as UNIX, Novell, and VMS. We have attempted to address this issue by designing AGUS, an acronym for "Account Generation Utility System". AGUS is a distributed, networked, multi-platform account generation system which requires no administrator intervention during normal operation. It is currently in active use at our site, and handles account generation for several UNIX workstation and Novell-based PC/Mac networks, as well as a VAX running VMS. As of this writing, it has been used to create over 15,000 accounts.

Our Environment

The University of Maryland, Baltimore County (UMBC) is a medium sized university of about 12,000 students. Our computing resources consist of several hundred UNIX workstations (predominately Silicon Graphics and Sun), and around 2000 PC and Macintosh machines. These are spread out across about 25 subnets. Most systems are centrally administered by the University Computing Services (UCS) department; some, although few, are autonomously managed by the department or research group that owns them.

University Computing provides about 85 single-user Silicon Graphics (SGI) workstations, two multi-user SGI systems, 600 PC/Mac machines, and a Digital VAX 4000/500 for general use by anyone registered with the University. For faculty and research use, we provide an SGI Crimson, a MIPS R8000-based SGI Challenge "M", and a 20-processor SGI Challenge "XL" system. AGUS handles all account generation on these systems. Currently, about 7000 of our students have accounts on our UNIX network and on the VAX, and about 10,000 have Novell-based accounts which are required for access to our PCs and Macs.

Each person is assigned a unique username based on his or her real name. Accounts stay active until the user leaves the University, or we terminate it for other reasons (disciplinary, etc.). Account information is kept online using the CCSO Nameserver software from the University of Illinois[1]. User authentication on the UNIX systems is handled via Kerberos release IV[2], from MIT Project Athena[3].

How we used to do things

Up until the fall semester 1993, we assigned student accounts on a course by course basis. Instructors would request accounts for their classes, and we generated a set of accounts for each course. Accounts were generated and distributed to instructors several days before classes began. Usernames were formed by taking the course name and appending ascending numbers to the end. Instructors handed the accounts out in class, and the accounts were deleted after final grades were due at the end of the semester.

This method had several disadvantages, the main one being that accounts were difficult to trace. Some instructors kept logs of which students had which accounts, but most didn't. If we got a complaint about an account being abused, it was very difficult if not impossible to trace the account back to a particular person.

These accounts also required a lot of work at the beginning of each semester. We had to solicit account requests from instructors, generate the accounts, print account information, and distribute accounts back to instructors. Invariably, some instructors would put in a late request for accounts, requiring us to repeat the process. We ended up spending way too much time generating accounts, when there were much more important things that needed to be done to prepare for the semester.

Class accounts were also unpopular with students. Many students received multiple accounts because they were enrolled in more than one class. The account names were difficult to remember because they were not based on the student's name. Students were unable to keep the same email address from semester to semester, and were forced to download files they wanted to keep to floppy at the end of each semester.

Our Account Generation System Requirements

What we wanted to provide to users

We wanted to enable any University-affiliated person to register for and receive an account on our Unix, VMS, and/or Novell networks. The registration process should be as quick and painless as possible, and unless absolutely necessary, it should not require the user to interact with any third party (i.e. instructors or University Computing staff). The accounts should stay active for the user's entire stay at the University. Account names should be easy to remember as well; preferably they should be based on the user's real name.

Obviously, temporary class accounts didn't fit our vision of what we wanted to offer to users; so, we decided to stop using them after the Spring Semester 1993. During that summer we began to consider other schemes for generating accounts.

Other Requirements and Desired Features

The first step in designing an account generation system was to identify the most important features it should have. We decided that in order to be useful to us, our system must have the following features:

Ease of Administration

It is essential that our system require as little administrator intervention as possible. During the first week of each semester, we will receive as many as 1000 requests for new accounts, and a steady stream of requests over the course of the semester. Manually processing these requests would be extremely tedious, repetitive and error-prone. Therefore, we designed AGUS to run completely unattended. During normal operation, the system runs itself and the administrator doesn't need to do anything at all.

Scalability

The system should be able to scale to accommodate a network of any size. As a medium sized University, we have an average user base of around 9000 active users spread out over several administrative networks. We designed AGUS to be capable of dealing with networks many times larger than ours.

Flexibility

One of our most important requirements, and a feature that sets AGUS apart from many similar systems, is complete platform independence. The system should be able to deal with any type of computer system that is capable of being networked via IP. We wanted to use the same system to create accounts on UNIX, VMS, and Novell based networks. The system should also be designed in such a way that it is simple to add additional system types to the configuration. For example, if the University decides to support user accounts on HP MPE systems, it should be relatively easy to extend AGUS to handle account creation under MPE.

Robustness

An account generation system should provide a good degree of robustness and error recovery. If it encounters a fatal error, it should notify the administrator. It should never leave an account in a partially-generated state.

Things We Considered and Tried

Before designing AGUS, we considered several other methods for generating accounts. Each has its own merits, but fails to satisfy one or more of our requirements.

Vendor Tools

The simplest way to create accounts is to use whatever tool the vendor provides. Silicon Graphics, for instance, provides a very nice GUI-based tool for account generation on Irix systems[4]. Unfortunately, this tool must be run manually for each new user to be added. In a typical fall semester at UMBC, over 2000 freshmen will request computer accounts. Using a GUI-based tool to individually add 2000 accounts would be hopelessly tedious.

Locally Developed Account Generation Scripts

It is fairly easy to develop a shell or Perl[5] script which handles account generation. An advantage of this approach is that the script can be tailored to do site-specific things. Most vendor tools do not provide this kind of flexibility. However, this approach was not automated or flexible enough to fit our needs. We wanted something that wouldn't require the staff to do repetitive work, and we wanted a system that ran on many different computer platforms.

Moira

Moira[6] is used at MIT to handle account creation on their Project Athena systems. We looked at this and found that, although it would do some of what we needed, it does not extend well to non-UNIX platforms, and it requires the presence of other Project Athena packages for proper operation. It also requires a commercial database package, which we did not want to use; one of our goals was to develop our system completely around public domain tools, so it would be of use to as many people as possible. Moira was also somewhat of an overkill for us. It was designed as a general purpose

service management system, and account generation is only a subset of its overall functionality.

AGUS

In designing AGUS, our initial goal was to create a system which supported automatic account generation and ran on every platform we supported. After the basic framework was in place, we refined the system to make it more robust and flexible. Now that it is stable, we are working to make it sufficiently generic that it will be useful to other sites with little or no modification.

AGUS Development History

AGUS, Take 1

AGUS was originally designed and implemented by a group of students as a class project for a Software Engineering course[7]. This implementation, which I'll call AGUS-1, was well thought-out and well-documented, and served our purpose for two semesters. Unfortunately, it had several critical design flaws which required us to eventually redesign it from the ground up.

AGUS-1 was implemented primarily using Bourne shell[8] scripts, and all processing was done on one of the multiuser student login machines, which was often overloaded. The system did not scale well at all. The machine would slow to a crawl when more than about 10 people tried simultaneously to register for accounts; no one would be able to get any work done. We ended up placing limits on how many people could register at one time, which greatly inconvenienced students, especially at the beginning of semesters.

After the first week of using AGUS-1, it became apparent that the system wasn't working too well; so, we decided to redesign it. Our goal was to provide the same menu-based interface to users, but eliminate the problems with the underlying implementation.

The New AGUS model

In redesigning AGUS, we worked to create a system with no major bottlenecks. A major goal was quick response time to users during the online registration process, even when several people are registering at the same time. We accomplished this by splitting AGUS into separate modules.

We initially coded most of AGUS in Perl version 4, with small parts of it in C. Developing and coding things in Perl is fast and straightforward, so we were able to get the initial system online quickly. However, Perl is difficult to extend, and we were forced to add extra modules to interface to KerberosIV and the CCSO database. It's also somewhat awkward to deal with complex data structures in Perl. Because of this we began to port the entire system to C. AGUS in its current state is about 90% C and 10% Perl.

AGUS From the User Perspective

AGUS presents users with a menu-based online registration system. A user walks up to an unused workstation or `telnets` to one of our multiuser login servers, and enters `register` at the username prompt. The user is then prompted for an identification (ID) number.[1] If the supplied ID number is valid, the system then displays a list of accounts for which the user is eligible to register.

Once the user has selected one or more accounts, the system generates a printout which shows the username and password for each account, as well as a form which explains our acceptable usage policy. The user presents valid identification to the print dispatch operator and signs the form. After 12 hours, the accounts are ready for use. Normally, this process requires no intervention by system staff.

What AGUS Doesn't Do

AGUS does not generate usernames or assign user identifiers. It requires that this information be present in the external user database. We did things this way for a couple of reasons; first, we didn't want to impose our naming scheme on other sites that want to use AGUS. Also, usernames and user identifiers are system-specific entities. For example, Digital's VMS uses two numbers to identify each user. These numbers are known collectively as the user's UIC (User Identification Code). UNIX uses a single number which it designates as the UID (User IDentifier). If AGUS were to generate user identifiers internally, it would have to know the rules for doing this on each supported system type, and it would have to be modified each time someone added a new system type. Since this goes against our system-generic design philosophy, we left it out of AGUS. Usernames and user identifiers are generated ahead of time, at the same time we load new student information into our database. One consequence of this is that we must assign usernames and identifiers for every registered student, not just the subset of students that will be using our computer systems. We have not found this to be a problem.

AGUS Requirements

We tried to design AGUS to require very few system resources and as little external software as possible. The existing implementation has minimal hardware requirements and requires an external database package capable of performing simple queries and updates.

[1] We currently use the student's Social Security Number for identification; however, the University has plans to begin issuing everyone a unique Personal Identification Number (PIN). If this ever happens, we'll probably start using the student's PIN rather than Social Security Number.

System and Hardware Requirements

The AGUS system requires a UNIX based system to run a single daemon process and a queueing system. The daemon accepts and queues new account requests, and the queue processor runs at periodic intervals and creates accounts. The daemon is *lightweight;* that is, it has negligible memory and CPU requirements. It can run standalone or via *inetd*[9]. The queueing system uses very little memory and requires enough disk space to hold the new account request queue. There is one queue entry per pending account request; each entry is stored in a separate file. Queue files are about 50 bytes long.

At UMBC we run the AGUS daemon and queue on a DEC 5000 with 96meg of memory. This system handles several other services such as e-mail relaying, backup DNS, and system backups. The AGUS system does not noticeably affect performance of the DEC.

AGUS also places a small load on the systems on which it is creating accounts. This involves things like modifying password files, creating home directories, etc., and is to be expected. In some cases (see "AGUS Implementation Details"), an additional daemon process handles account creation.

The External Database

AGUS requires read and write access to an external database package to keep track of users. The database must be able to do searches using a user's ID number as the key. For any given user, the database must be able to return four different fields: the user's real name; a unique username and identifier for each type of account the user is registering for; and a *group* which specifies which accounts the user is eligible to receive. The database must also keep tabs on accounts the user already has, so that people cannot register for the same account more than once. A sample database entry is shown in Figure 1.

```
id:              999999999
name:            bernie freeman
unix_username:   bfree
unix_uid:        451
vms_username:    bfreeman
vms_uic:         451.100
group:           research
inst_created:    1994/04/05
```

Figure 1: Sample database entry

We use the CCSO nameserver as our external database. We chose this package because it was free and satisfied other requirements we had (unrelated to AGUS). AGUS was written so that it should be easy to adapt it to use any database system that includes a programmatic interface. It includes a small database system (based on NDBM[10]) for people who wish to try AGUS out without installing CCSO or adapting AGUS to use a different package.

AGUS Naming Requirements

AGUS requires that each computer network it handles be given a unique name. The system uses these names internally and when prompting users for account types. At our site, our general-use SGI UNIX network is designated *instructional*; our faculty and research UNIX network is called

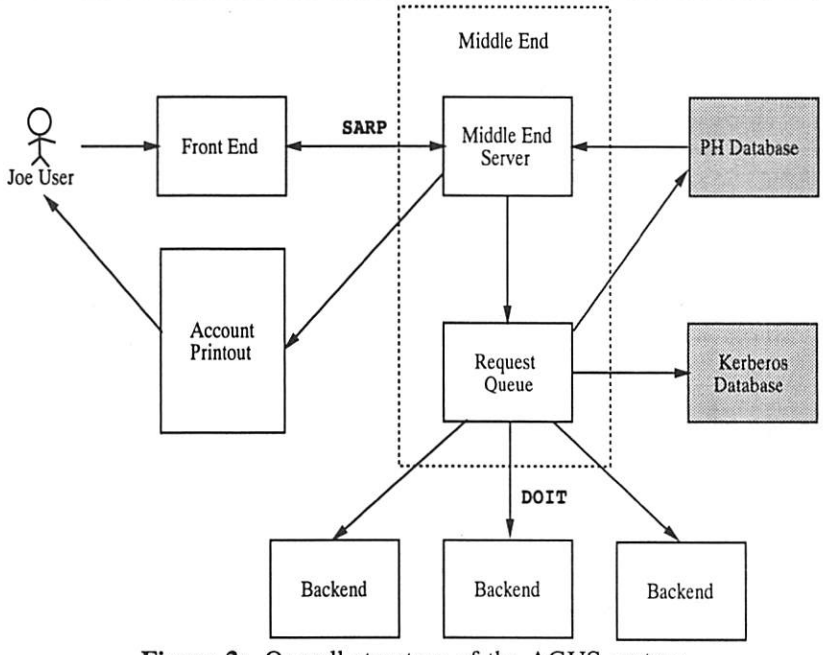


Figure 2: Overall structure of the AGUS system

research; and our VAX system goes by *vms*. Think of this name as being similar to a NIS[11] domain name; it refers collectively to all of the computers in a single administrative domain.[2] You should choose names which are descriptive enough that a new user will understand which computers each name refers to.

For each computer network, AGUS uses three database fields for a given user. The first specifies the user's username on the network. The second field is a unique identifier for the user; under UNIX this is the user's UID, and under VMS it is the UIC. Under Novell, identifiers are managed internally so the field is not used. The third field specifies the account creation date, and is set at the time AGUS generates the user's account. For example, referring back to Figure 1, Bernie Freeman's UNIX username and UID are determined by his `unix_username` and `unix_uid` database entries. The `inst_created` field shows the date that Bernie registered for an account on our instructional UNIX network. There is a similar field, `res_created`, for our research network. Bernie doesn't have a `res_created` field, which means he has never registered for a research account.

AGUS determines which database fields to use by consulting a configuration file, `agus.conf`. This file is presented in the next section.

AGUS Implementation Details

The overall structure of the AGUS system is shown in Figure 2. System-specific, time-consuming tasks such as creating home directories and assigning disk quotas are wrapped into separate *back end* modules. When a user logs in as `register` to request a new account, he interacts with a program we call the *front end*. The front end prompts the user for an ID number and passes it on to the next layer of AGUS, the *middle end*. The middle end validates the user's ID number and determines what accounts the user is eligible to receive, if any, based on the user's group field in the external database. The user chooses one or more account types. The middle end then generates initial passwords, creates an information printout for the user to pick up, and places the registration request into a queue. At periodic intervals (every 6 hours at our site), the system processes the request queue and creates each account by sending commands to the appropriate back end module for that account type. It also makes updates to the external database and, when necessary, the Kerberos database.

AGUS learns about each computer network by reading a configuration file called `agus.conf`. This file contains one line for each computer network. Lines consist of several colon-separated fields. Figure 3 shows a sample `agus.conf` file. Lines which are too long for the page are split with a backslash ('\').

```
instructional:ds1.gl.umbc.edu:\
  umbc8 and umbc9:gl.umbc.edu:\
  inst_created:kerberos:\
  unix_username,unix_uid,group,name
research:umbc7.umbc.edu:umbc7:\
  research.umbc.edu:\
  res_created:kerberos:\
  unix_username,unix_uid,group,name
vms:/umbc/agus/bin/vms-rcp:umbc2:\
  umbc2.umbc.edu:vms_created:plain:\
  vms_username,vms_uic,group,name
```

Figure 3: Sample `agus.conf` file

The first field in each line is the name of the network. In this example we have entries for three different networks: *instructional*, *research*, and *vms*.

The second field contains information about the AGUS backend for this network. It specifies either a hostname or a pathname. See the "Backends" section for more on this field.

The third and fourth fields contain information to display on the printout that the user picks up after registering. The third field is the hostname of a computer (or computers) which the user can access (via `telnet` or `rlogin`) for remote access to his account. The fourth field is the electronic mail address of the network, and is used to tell the user his email address.

When AGUS creates an account, it stores the creation date of the account into AGUS's external database. The fifth field of the `agus.conf` file specifies which database field to use to store this information. For example, when creating a research account, AGUS should store the account creation date into the `res_created` field of the new user's database entry. For instructional accounts, we use `inst_created`, and VMS accounts use `vms_created`.

The sixth field contains information about how passwords are handled on this network. We currently support three different methods: *kerberos*, for passwords stored in an external Kerberos database; *crypt*, for standard one-way UNIX password encryption[12]; and *plain*, for plaintext passwords (required in some non-UNIX environments).

The seventh and final field consists of several comma-separated words. Each word is the name of a field in AGUS's external database. These fields are used to create an entry for a new account on this network in AGUS's account request queue. See "The AGUS Request Queue" for further information about the queue.

[2]Note that for security reasons, you should choose a name that is different from the NIS domain name, if you run NIS at all.

AGUS uses a configuration file called `group.conf` to determine for which accounts a user is eligible to register. A sample `group.conf` file is shown in Figure 4.

```
staff:instructional,research,vms
systems:instructional,research,vms
research:instructional,research,vms
general:instructional,vms
guest:instructional
inactive:
```

Figure 4: Sample `group.conf` file

The `group.conf` file contains one line per group. Each line consists of a group name, a colon separator, and a list of account types that users in the group can register for. For example, someone in the *general* group can register for an instructional account or a VMS account. If a user is unlucky enough to be in the *inactive* group, he cannot register for any accounts at all. Looking back at Figure 1, we see that Bernie's group is *research*; this entitles him to an instructional, a VMS, or a research account.

Interaction Between the Front End and Middle End

The front and middle end modules communicate over TCP using a simple 7-bit ASCII protocol similar to SMTP[13]. We call this protocol SARP (Simple AGUS Registration Protocol). SARP provides commands to query the database for a specified ID number, select one or more account types, and commit registration requests to the registration queue. The full protocol specification is provided in Appendix A. The advantage of this approach is that we can develop front end clients which run on a wide variety of hardware and support several different user interfaces. Our current client uses the UNIX curses library, and we have started work on clients based on Motif[14], TCL/Tk[15], and World Wide Web CGI[16] scripts. We are also developing an alternative UDP-based server/client protocol, which would be more appropriate for stateless clients such as WWW browsers.

The AGUS Request Queue

AGUS's account request queue is similar to the mail queue in sendmail[17]. We considered creating accounts at the same time users registered for them, but quickly abandoned this idea because it would create an undesirable load on the backend server hosts when multiple people registered simultaneously. Our solution was to place account requests into a queue, and then process them one-by-one at specific times via *cron*[18].

The queue consists of one directory for each type of account. At our site there are directories called `instructional`, `research`, `vms`, and `novell`. These directories contain one file per pending account request. Files are named after the registrant's ID number. As an example, let's say that Bernie Freeman from Figure 1 decides to register for a research account. Assuming the AGUS queue is rooted at `/agusq`, the pathname of Bernie's queue file would be `/agusq/research/999999999`.

Queue files are fairly simple. They contain all the information the backend needs to create the account, with each piece of information on a separate line. The `agus.conf` file (see Figure 3) specifies the actual information that goes into the queue file. The first line of the queue file is always the user's password, encoded according to the information in the sixth field of the network's `agus.conf` entry. For Kerberos based accounts, the password is set to '*' and a separate routine updates the Kerberos database. For most other UNIX accounts, the password is stored encrypted in the queue file; for most non-UNIX accounts it is stored in plain text.[3] On UNIX systems, the backend stores the password in `/etc/passwd` exactly as it appears in the queue file.

The rest of the queue file lines consist of information from the external database. The seventh field of `agus.conf` contains a list of database fields; AGUS looks up each of these fields and writes its value to the queue file, one entry per line.

To illustrate, let's again assume that Bernie Freeman wants to register for a research account. According to `agus.conf`, the research network uses Kerberos to store passwords, and the database fields to use for the queue file are `unix_username`, `unix_uid`, `group`, and `name`. Based on this information, Bernie's queue file will look something like this:

```
*
bfree
451
research
bernie freeman
```

A cron job runs at specific intervals (every six hours at our site) and scans the queue directories for requests. For each request it finds, it sends the request to the appropriate backend server. The backend server returns a result code which says whether the account creation was successful. If so, the request is removed from the queue; if not, it is retried the next time the queue is processed via *cron*. Requests that fail more than once are reported to the system administrator.

[3]Cleartext passwords are a security risk on public networks and should be avoided when possible; we are working on a way to handle these securely which is easily portable to non-UNIX platforms.

Backends

Backend modules are responsible for actually creating new accounts. For example, on a UNIX system, the backend program would be responsible for creating password file entries, assigning and creating home directories, setting permissions, assigning quotas, and setting up initial startup files (`.cshrc`, `.login`, etc.). For these types of systems, the backend program runs on the system where this type of work is typically done, e.g. the NIS master server for networks that run NIS. The middle end communicates with backends using one of two different methods: the DOIT network protocol or an external backend program. AGUS determines which of these methods to use by looking at the second field of `agus.conf`. If the field starts with '/', AGUS assumes that it specifies a pathname to an external program and attempts to execute the program. If the field does not start with '/', AGUS treats it as a hostname and attempts to connect to the host using the DOIT protocol.

The DOIT Protocol

DOIT is a 7-bit ASCII protocol similar to SARP. Our UNIX-based backend program speaks the DOIT protocol. The protocol specification is provided in Appendix B. DOIT provides a command to create an account given several different colon-separated parameters (typically username, encrypted password, UID, group name, and full name). AGUS obtains these parameters by reading the queue file and massaging the contents into the proper format. A client can generate as many accounts as it wishes during one session.

External Programs

Designing a backend that speaks an ASCII protocol over the network typically requires TCP/IP connectivity and some sort of BSD-socket-like programming interface. We realized that it if this were a requirement, it would be difficult to develop backends for non-UNIX platforms. For these types of systems, AGUS can invoke an external program to communicate with the backend. This is similar to the behavior of sendmail's *prog* mailer. For example, to generate accounts on our VMS system we invoke an external program which uses *rcp*[19] to copy the AGUS queue file over to a special account on the VAX. Every several hours, the VAX runs a script that checks for new files in that account, and creates accounts based on what it finds. We use a similar program for our Novell systems.

Examples

A few examples should help illustrate how everything fits together. For these examples, let's assume that our old friend Bernie Freeman wants to register for an research UNIX account and a VMS account. He would first walk up to an unused workstation and log in as `register`. The login shell for the *register* account is an AGUS front end client, which prompts him for an ID number. The system then connects to the AGUS middle end server, which determines whether the ID number is valid. If so, the middle end returns a list of account types Bernie is eligible to receive. Since Bernie is in the *research* group, he is automatically eligible for an instructional UNIX account, a VMS account, and a research UNIX account. Since he already has an instructional account, though, he can only register for research or VMS. The front end formats this response and presents it to Bernie as a menu. Bernie is interested in a research and a VMS account, so he selects these from the menu. The system sends this information to the middle end server. The middle end generates random passwords for each account, creates two queue files, and generates a printout of account information for Bernie to pick up.

The SARP transaction between the front and middle ends for our example is shown in Figure 5. Input from the front end is shown in **bold** typeface and responses from the middle end are shown in `plain` typeface. Refer to Appendix A for a description of the individual SARP commands.

```
220 Server ready.
IDNO 999999999
210-bernie freeman
210-research
210 vms
TYPE research
200 TYPE research OK
TYPE vms
200 TYPE vms OK
RGST
250 Request queued
QUIT
221 Goodbye!
```

Figure 5: Sample SARP Transaction

```
220 Server ready.
DOIT bfree:*:451:research:bernie freeman
200 Okay
DONE
221 Goodbye!
```

Figure 6: Sample DOIT Transaction

The system creates Bernie's accounts the next time it processes the AGUS request queue. For the UNIX account, it establishes a connection to the backend server running on the NIS master server of our research UNIX network (specified in the second field of `agus.conf`), and creates the account using the DOIT protocol. Figure 6 shows the DOIT transaction. For the VMS system, the system executes a Perl script called `vms-rcp`. The script copies the queue file over to the VAX, where the account is created via a cron script. Note that AGUS does not

assign home directories; this is handled internally by the backend program.

Conclusion

Although there were some growing pains, AGUS has proven to be a good solution for automatic account generation at our site. Users like the consistent interface it provides, and the system staff appreciate its completely automatic operation. In addition, it is the only system we know of that can be extended to many different OS platforms, both UNIX and non-UNIX. By making it freely available we hope it can be of use to others as well.

Availability

The AGUS system has been in production at UMBC for almost two years now. It is not currently release ready, but we are actively working on preparing it for public beta testing. We hope to have something to offer within the next several months.

Author Information

Paul Riddle is a Systems Programmer for Hughes STX Corporation, working as a contractor at NASA/Goddard Space Flight Center in Greenbelt, MD. Paul wrote the AGUS registration server and queue processor. Paul did most of his work on AGUS while an employee of the University of Maryland, Baltimore County, and now maintains the code in his spare time. This is his second LISA Paper.

Paul Danckaert is a Systems Programmer for University of Maryland, Baltimore County. Paul wrote the AGUS front end interface, and helped design the basic AGUS model and protocol specification. His main area of research is in computer security, and his most recent project enhanced IRIX modules for Rscan, a security scanner by Nate Sammons from Colorado State. This is his first LISA paper.

Matt Metaferia currently works at MCI Telecommunications in Washington, D.C. Matt designed and coded the UNIX based backend server and account generation engine. Matt did all of his work on AGUS while employed at the University of Maryland, Baltimore County.

Appendix A – Simple AGUS Registration Protocol (SARP) Specification

Overview

SARP is a network protocol similar to SMTP and FTP[20]. It runs over TCP and provides a simple mechanism for users to register over the network. Any client that can speak SARP can be used for registration.

SARP runs as a TCP service on the well-known port specified by the *agus* entry in the `/etc/services` file. Currently it runs from port 293. It should always run from a privileged port (port number less than 1024) so that unprivileged users can't create a "phony" SARP service in the event that the real one dies.

Connection Negotiation

When the SARP server receives a new connection, it should issue a greeting banner and begin speaking the SARP protocol. All server responses must be preceded by a 3-digit numeric result code. The greeting banner should include a `220` result code, indicating that the server is ready to register a new user.

Server Responses

Each response consists of a 3-digit number followed by a text explanation of the response. Example:

```
200 Command succeeded
```

A hyphen immediately following a response number indicates a multi-line response. The last line in the response will not have a hyphen. A sample multi-line response follows:

```
210-bernie freeman
210-instructional
210 vms
```

Numeric responses each have certain meanings. Clients can parse these responses to determine whether commands succeeded or failed. The meanings are similar to what they would be in SMTP or FTP:

`2xy` Indicates general permanent success.
`4xy` Indicates general transient failure.
`5xy` Indicates general permanent failure.

`x0y` A syntax related message.
`x1y` An informational message.
`x2y` A message related to the connection.
`x3y` An authentication related message.
`x4y` Unspecified.
`x5y` A message related to the registration server.

Here are the actual response codes used by the SARP and DOIT protocols, in numeric order.

`200` – Command succeeded.
`210` – Successful informational response.
`220` – Server ready for new session.
`221` – Closing control connection.
`401` – Transient error in command parameters.
`450` – General transient failure.
`500` – Unrecognized command.
`501` – Error in command parameters.
`503` – Improper order of commands.
`504` – Command not implemented for this parameter.
`520` – Closing control connection due to fatal error.
`530` – ID number not found in database.
`550` – Requested action not taken (failed).

Server Commands

Commands are four letter words or abbreviations. All input is case insensitive; `IDNO` is the same as `idno`, `iDnO`, etc.

Here is the canonical list of SARP commands and what they do.

IDNO – Specify an ID Number for Registration

Syntax: `IDNO` *id-number*

This specifies that a person with ID number *id-number* wishes to register for an account. The server should look up the ID number in its database. The ID number must be a 9-digit number; if it's not, the server should return a `501` response (parameter error).

If the ID number is valid and present in the database, the server should return a multi-line `210` response. The first line prints the registrant's name, and each succeeding line lists a valid account type for the registrant.

A valid account type is one that the user (1) is allowed to register for, and (2) has not yet registered for. If the user is valid but has already registered for all allowed accounts, the server should still return a `210` response, but should not list any account types with the response.

An `IDNO` command may not be specified more than once. Duplicate `IDNO` commands should return `503` (sequence error) responses.

Examples

Sample `IDNO` responses follow:

```
210-david duffy
210-instructional
210 research
```

This user is in the database and is allowed to register for either an instructional or a research account. The user is either not allowed to register for a VMS account, or has already registered for one.

```
530 IDNO 123456789 not present\
  in database.
```

This user is not in the database.

TYPE – Specify an account type

Syntax: `TYPE` *account-type*

This command selects an account type to register for. It must follow an `IDNO` request; if it doesn't, it should return a `503` response. It requires one argument; if no arguments are specified, it should return a `501` error.

The argument to `TYPE` must be one of the responses generated by the `IDNO` command. Invalid responses should return `504` (command not implemented for parameter) responses.

A user may register for a particular type of account only once in a session. Multiple attempts to register for a single account type, should result in `503` responses.

Assuming the requested account type is valid, the `TYPE` command should return a `200` response.

Examples

Sample `TYPE` responses follow:

```
200 TYPE instructional OK
```

The user wishes to register for an instructional account, and is authorized to do so.

```
504 Invalid account type
```

The user tried to register for an account type not returned by the `IDNO` command.

RGST – Register for Account(s)

Syntax: `RGST`

The `RGST` command actually registers the user after he or she has specified an `IDNO` and at least one `TYPE`. If the user attempts to `RGST` before doing either of these things, the server should return a `503` response.

If the registration succeeds, the server returns a `250` response; otherwise it returns a `550` response with the reason included in the text of the response.

`RGST` should do an implicit `PRNT` operation when it succeeds.

PRNT – Print Account Registration Information

Syntax: `PRNT`

This command prints registration information for the user. The purpose of this command is to reprint the information in case it didn't print the first time. `RGST` should handle printing for new registrants.

`PRNT` must be specified after the user enters an `IDNO`, but before they enter a `TYPE`. Any other usage should return a `503` result code.

If the print job succeeds, `PRNT` should return a `250` response.

QUIT – Exit From the Registration Server

Syntax: `QUIT`

`QUIT` is used to disconnect from the server. All pending action is cancelled with the exception of queued account requests. `QUIT` should return a `221` response code before dropping the connection.

Appendix B – DOIT Backend Registration Protocol Specification

DOIT is a network protocol similar which works similarly to SARP. DOIT runs as a TCP service on the well-known port specified by the *agus-doit* entry in the `/etc/services` file. Currently it runs from port 294.

DOIT only needs to be able to handle a single connection at a time. Clients should take pains to ensure that they only have one active connection active at any given time. This restriction allows the

backend AGUS software to avoid dealing with mutual exclusion, file locking, and race conditions.

Connection Negotiation

When the DOIT server receives a new connection, it should issue a greeting banner and begin speaking the DOIT protocol. The greeting banner should include a `220` result code, indicating that the server is ready to accept requests.

Server Responses

DOIT server responses follow the same conventions as SARP responses, including a 3-digit numeric response code and a text message. For more details, please refer to Appendix A.

Server Commands

The DOIT protocol supports only two commands: `DOIT` and `DONE`.

DOIT – Specify Account Registration Data

Syntax: `DOIT` *username:password:uid:group:gecos:quota*

This specifies data for an account to be created. The DOIT protocol requires that six fields be specified; however it does not do syntactical checks on the individual fields. That is left up to the backend. The backend is responsible for generating all information (home directories, shells, etc) not specified via `DOIT`.

Once the backend receives a `DOIT` command, it should create the account on the fly and return a response code depending on whether or not it succeeded. A `200` response indicates that the request succeeded, and that the client may assume that the account has been successfully created. The client need not queue or retry the request after a `200` response.

A `450` response indicates a transient failure. This means that the backend failed for the current request, but the client may continue to send further requests during the current session.

Errors in the parameters should return `401` responses. These include things like improperly formatted usernames, invalid UIDs, nonexistent groups, etc. The backend should also notify the AGUS administrator of these errors, since they generally indicate some sort of internal problem.

Fatal errors should give a `550` response. This means that there is some sort of catastrophic problem with the backend, and the client should immediately issue a `DONE` and disconnect.

DONE – End a Registration Session

Syntax: `DONE`

The `DONE` request indicates that the client is finished sending `DOIT` requests. The server should perform any post processing and the return a `221` response to the client, which causes the client to disconnect. Typical examples of post processing include generating NIS maps, rebuilding quotas, etc.

Note that it is the backend's responsibility to ensure that post processing succeeds. The client should not requeue requests if post processing fails. Therefore, the server should return a successful response whether the post processing succeeds or not.

References

[1] Dorner, S., and Pomes, P., *The CCSO Nameserver – A Description*, 1992.

[2] Steiner, J., Neuman, C., and Schiller, J., *Kerberos: An Authentication Service for Open Network Systems*, Proc. USENIX Winter Conference, January 1988. 30, 1988

[3] Steiner, J., and Geer, D., *Network Services in the Athena Environment*, July 21, 1988.

[4] "cpeople(1) Manual Page," *IRIX Reference Manual*, Silicon Graphics, 1993.

[5] Schwartz, R., and Wall, L., Frogramming Perl, O'Reilly and Associates, 1991.

[6] Rosenstein, M., Geer, D., and Levine, P., *The Athena Service Management System*, 1991.

[7] O'Hern, T., *AGUS Account Generation Utility System*, UMBC, 1993.

[8] "sh(1) Manual Page," *IRIX Reference Manual*, Silicon Graphics, 1993.

[9] "inetd(1M) Manual Page," *IRIX Reference Manual*, Silicon Graphics, 1993.

[10] "NDBM(3B) Manual Page," *IRIX Reference Manual*, Silicon Graphics, 1993.

[11] Sun Microsystems, *Network Information Services*, 1991.

[12] "crypt(3C) Manual Page," *IRIX Reference Manual*, Silicon Graphics, 1993.

[13] Network Working Group RFC 821 "Simple Mail Transfer Protocol," Postel, J., 1982.

[14] Open Software Foundation, *OSF/Motif Programmer's Reference*, Prentice-Hall, 1991.

[15] Ousterhout, J., *Tcl and the Tk Toolkit*, Addison-Wesley, 1994.

[16] "The Common Gateway Interface," URL http://hoohoo.ncsa.uiuc.edu/cgi/overview.html.

[17] Allman, E., *Mail Systems and Addressing in 4.2bsd*, January 1983.

[18] "cron(1M) Manual Page," IRIX Reference Manual, Silicon Graphics, 1993.

[19] "rcp(1C) Manual Page," IRIX Reference Manual, Silicon Graphics, 1993.

[20] Network Working Group RFC 959 "File Transfer Protocol", Postel, J., and Reynolds, J., 1985.

OpenDist – Incremental Software Distribution

Peter W. Osel and Wilfried Gänsheimer – Siemens AG, München, Germany

ABSTRACT

OpenDist provides efficient procedures and tools to synchronize our software file servers. This simple goal becomes challenging because of the size and complexity of supported software, the diversity of platforms, and because of network constraints.

Our current solution is based on *rdist*(1) [1]. However, it is not possible anymore to synchronize file servers nightly, because it takes several days just to compare distant servers.

We have analyzed the update process to find bottlenecks in the current solution. We measured the effects of network bandwidth and latency on *rdist*. We created statistics on the number of files and file sizes within all software packages.

We found that not only the line speed, but also the line delay contributes substantially to the overall update time. Our measurements revealed that adding a compression mode to *rdist* would not have solved our problem, so we decided to look for a new solution.

We have compiled a list of requirements for evaluating software distribution solutions. Based on these requirements, we evaluated both commercial and freely available tools. None of the tools fulfilled our most important requirements, so we implemented our own solution.

In the following we will describe the overall architecture of the toolset and present performance figures for the distribution engine that replaces *rdist*. The results of the prototype implementation are promising. We conclude with a description of the next steps for enhancing the OpenDist toolset.

Our Environment

The CAD Support Group of the Semiconductor Division of Siemens AG installs, integrates and distributes all software needed to develop Integrated Circuits. We have development sites in Germany (München and Düsseldorf), Austria (Villach), the United States (Cupertino, CA), and Singapore. The development sites are connected by leased lines with a speed of 64 to 128 kBit/s. At each site, a central file server stores all software. Client workstations mount software from these servers. Software is installed and integrated in München and distributed to all other development sites. System administrators of the development sites initiate the transfer on the master server in München.

The CAD Support Group takes care of the CAD software and tools, only. A separate department is responsible for system administration, i.e., maintenance of the operating system and system tools, backups, etc.

Our software distribution problem differs in many ways from the one solved by traditional software distribution tools. Most software distribution tools we looked at are designed to distribute a moderate number of fairly static software packages of moderate size to many clients.

In contrast, we have to synchronize few file servers (under a dozen), which store many (about 200) packages of sizes ranging from tiny (a couple of kilobytes) to huge (1.8 GBytes). The total size of the software we store is currently 25 GBytes, 10-15 GBytes are currently being kept up-to-date at all sites. Many packages are changed each day. A change might update only a single file of a few bytes or could change up to 50,000 files for a total of 1 GBytes per day. Every month about 10 % of the software change. Most changes are small, but many files are constantly updated. The installation of a huge patch or a new software package changes many files at once.

There is no separate installation- or test-server, all changes are applied to the systems while our clients are using them. The changes are tested in München and, ideally, copied to all slave file servers within one day. Synchronizing or cloning file servers is the best way to describe our setup.

Our Current Solution

Our current software distribution process uses *rdist*(1) to find changed files and to update slave software servers. It is no longer possible to compare two software servers in one night. A complete check of all software packages on the slave file server in Singapore would take several days which is not acceptable nor feasible. During that time, software packages would be in inconsistent states, and changes of the master software server could take

up to a week to be transferred to the slave file server. Though it is possible to apply different update schedules – updating small packages daily, some weekly – the setup is not satisfactory. With an ever-increasing number of software packages and an ever-growing size of each software package, the distribution process using *rdist* is not acceptable any more.

Searching The Bottleneck

We have analyzed the update process to find bottlenecks in our current solution. We analyzed our lines and measured bandwidth, latency and compression rate (all leased lines are equipped with datamizers – devices that compress all traffic). We created statistics on the number of files and their size for more than 200 software and data packages. Commercial software packages, technology data and cell libraries, as well as many free packages like X11 and gnu tools were analyzed. We were also interested in the compression rate and time of software packages and how much the compression rate differs when software packages are compressed file by file or as a complete archive. We analyzed where *rdist* spends its time during updates. Compared to the installed software, our change rate is small, so finding changed files must be efficient. Changes can be rather huge, so the transmission of changed files must be efficient, too.

The Benchmark

We wrote a benchmark suite that measures the elapse time needed to perform typical software distribution operations such as installing, comparing, deleting, and updating files of different sizes, installing symbolic and hard links. All operations were executed many thousand times to equalize differences of the link performance.

The benchmark measures *ping*(1), *rcp*(1), and *rdist*(1) performance and times. Each *rdist* test runs on a directory with an appropriate number of random files of the same size. Each test contains an add, check, update and delete sequence. The file size is increasing from 1 Byte to 1 MBytes. Thus the effect of transfer rate and *rdist* protocol can be separated. The *rdist* part of the benchmark source tree contains approximately 5,000 files. This sums up to 10,000 transferred files, 5,000 check actions, 5,000 delete actions and 30 MBytes transferred data per test run. *rcp*(1) times are measured for a text, a binary and a compressed file of 1 MBytes each. This shows the achieved on-line compression.

The leased lines (except the dialup ISDN link) are shared by many users. So it is not astonishing that the benchmark results varied a lot, sometimes by more than a factor of three. To make our benchmark of the line performance more comparable, we calculated the average value for the best results of several runs of the benchmark. Some of the small numbers are within the magnitude of time resolution and must be interpreted cautiously.

The Results

Size and Composition

Software packages vary substantially in size and composition of file types, however bigger packages don't necessarily have bigger files, they have a few huge files, but the average file size is more or less independent of the total size of the package (Diagram 1).

	LAN	MÜNCHEN	DÜSSELDORF	VILLACH	CUPERTINO	SINGAPORE
Line Type	Ethernet	ISDN	X.25	leased	X.25	leased
Nominal Line Speed [kBit/s]	10,000	64	64	128	64	64
Transfer rate [kByte/s]	90-100	6-7	4-5	7-12	2-3	3-4
Ping Response Time [ms]	<1	33-88	188-372	81-311	530-1083	617-1375
rdist file create [s]	0.2	0.2	1.2	0.6	2.1	4.5
rdist file check [s]	0.02	0.06	0.5	0.2	1.	2.1
rdist file delete [s]	0.1	0.13	0.5	0.3	1.	2.3
10 kBytes transfer rate [kByte/s]	-	5.9	2.5	4.9	1.5	1.4
Run Benchmark [h]	1	2.5	7	4	12	24
rdist check SW subset [h]	-	-	16	8	-	>80[2]
OpenDist check SW subset [h][3]	0.5	-	2[1]	.5	.75	.75
rdist check all SW [h][2]	3	-	69	27	140	290
OpenDist check all SW [h]	1.5	-	5[1]	1.5	2	3

[1]Increased time, because software pools in Düsseldorf are accessed via NFS not UFS.

[2]Estimated.

[3]This subset consists of technology data and is changed and distributed daily. The subset contains approximately 150,000 files with a total of 1.1 GBytes.

Table 1: Line Characteristics

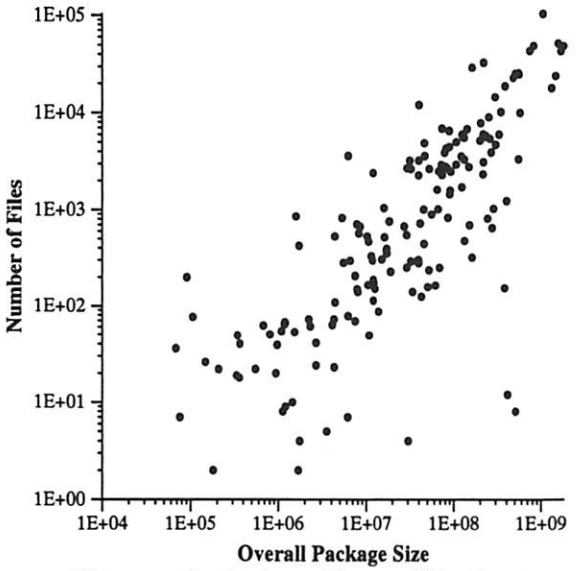


Diagram 1: Package Size vs. File Count

Compression Factor

The average compression factor of our software packages is three. Most of our software packages were compressed by this factor, though we observed compression factors between two and five.

When using *gzip*(1), you can regulate the compression speed between fast (less compression) and slow (best compression). For our software packages, increasing the compression quality reduces the compressed file size by less than 5 %, the compression time however sometimes increased by more than 200 % (Diagram 2). The default compression level of 6 is a good compromise, so we decided to use it.

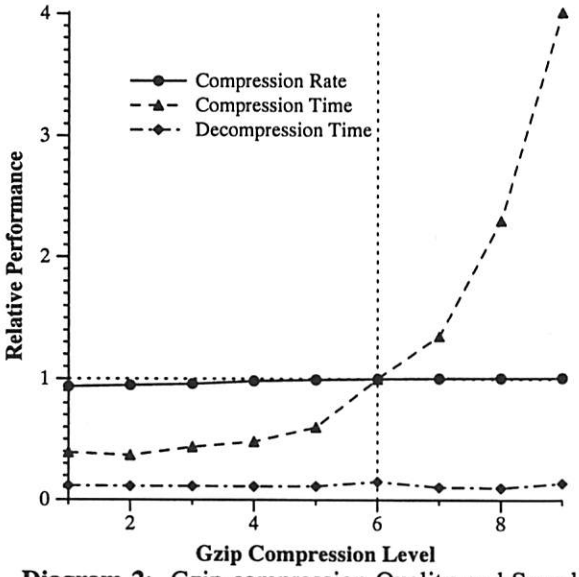


Diagram 2: Gzip compression Quality and Speed

Though our leased lines are equipped with datamizers that compress network traffic, it is worthwhile to compress archives before transmission. Datamizers increased the transmission rate of uncompressed data by 10..15 %, whereas *gzip* reduced the data to a third of their original size.

Compression Rate

On a SPARCstation 10/41 (Solaris 2.4, 128 MBytes memory) *gzip* created compressed data at a rate of 65 kByte/s, many times faster than the speed of our leased lines. This figure is important to know when you want to pipeline the creation, compression, and transmission of update archives. In case the throughput of the lines is in the same order of magnitude as the *gzip* output rate, it would be advisable to decrease the compression level.

Decompression

Decompressing the archives with *gunzip*(1) is usually six times faster than compressing the data. Decompression time does not depend significantly on the compression quality chosen for compression (Diagram 2).

Compression and Archives

It is better to compress an archive of files than to archive compressed files. Compressing complete packages is significantly faster and creates smaller archives than compressing each file separately and archiving the compressed files. For example, archiving and compressing X11R6 was completed in three minutes elapse time, and the overall size was reduced by 55 %. Compressing each individual file and archiving the compressed files in a second step took five minutes elapse time and reduced the overall file size by only 45 %. All tests were performed several times on an unloaded machine. Compressing individual files and archiving them needs many more file and disk operations compared to archiving the uncompressed files and compressing the archive. Compressing several small files (or small network packets) is not as efficient as compressing the files in a single run.

Transmission and Archives

It is better to transmit an archive of files than to transmit each file individually. Depending on the file transfer protocol used, the latency of the line has a high impact on transfer rates. The smaller the files and the higher the latency, the higher is the delay caused by inefficient protocols.

The latency increases the time *rdist* needs to check or create files. If you have many files, *rdist* needs a long time to compare master and slave server. If many or all files changed (e.g. when installing a new software package), *rdist* will need much more time to transfer all files. The average file size of our software packages is 30 kBytes (Diagram 3). To our Singapore site, we need about 10 seconds (3 kByte/s) to transfer a file of this size. However, *rdist* needs more than 4 seconds to create the new file, for a total transmission time of 14

seconds (40 % increase), a 30 % decrease in transfer rate. The transfer rate for 10 kBytes files is only half of the normally achievable transfer rate (See Table 1).

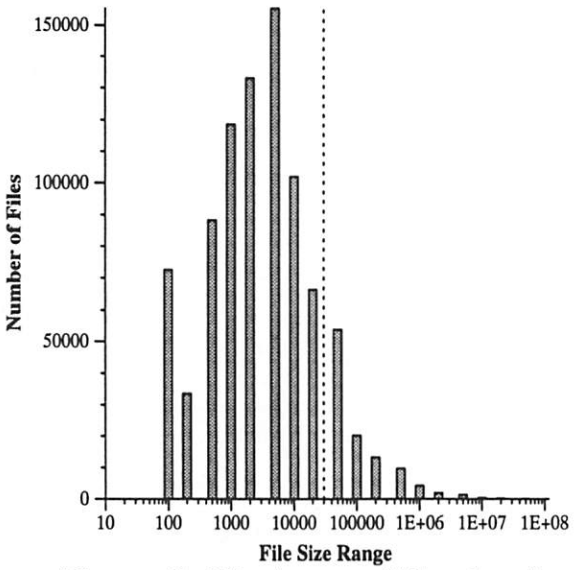


Diagram 3: File size range (All packages)

Besides avoiding protocol overhead, the transmission of archives has additional advantages. By first transferring all changed files to a holding disk, and installing changes locally on the remote server from the holding disk, the time during which the software package is in an inconsistent state is significantly reduced. Moreover, we can use the same tools to archive and roll-back changes. The installation of changes can be done asynchronously, so a system administrator at the remote site can easily postpone updates. The advantages compensate the disadvantage of needing holding disks to temporarily store the file archives.

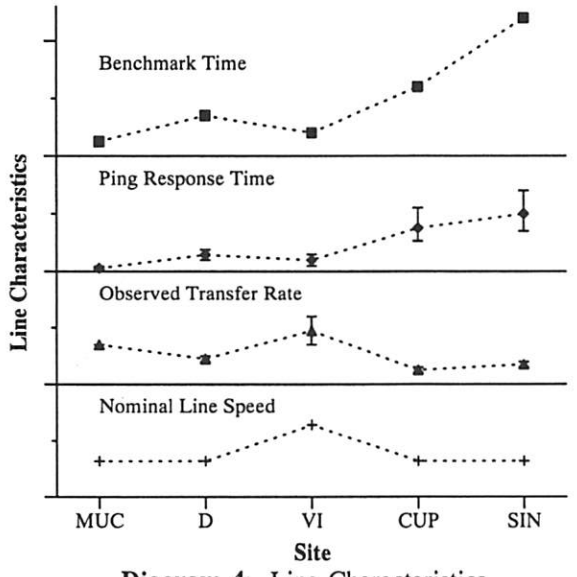


Diagram 4: Line Characteristics

rdist and Latency

Although the line speed from München to Villach and to Singapore differs by only a factor of two, the time needed to run the *rdist* benchmark differs by a factor of six (see Table 1 and Diagram 4). The ping response time (and therefore latency) has a greater impact on the time *rdist* needs to create or compare files than the line speed.

Benchmark Summary

Our measurements have revealed that the line speed is not the only bottleneck: the latency also plays an important role. *rdist* compares source and target directory file by file. Because the time for this is proportional to the latency, and because our change rate is small compared to the installed software, adding compression to *rdist* would not have solved our problem. *rdist* spent most of its time trying to figure out what to update, and not actually updating files. On the other hand, if a new version of our biggest software package is installed, we have to transmit 1.8 GBytes, so transmission must be optimized, too. The transmission of single files is another bottleneck as in our environment, the protocol overhead and transmission time are in the same order of magnitude, which reduces the average actual transfer rate by up to 30 %. For an efficient solution in our environment, files that have to be updated must be archived first and then be transmitted in one large file.

Upgrading our lines would not solve our problem, because the latency would not get small enough. It is also a very costly solution.

We found that we had to tackle two problems: making the finding of changed files more efficient, and making the transmission of data more efficient. We began to look for a new solution.

Requirements for Software Maintenance

We compiled a long list of requirements that a new solution should fulfill. Here are some of the more important ones:

Optimal Support of Incremental Distribution

We do not want to trace changes as they are applied and re-apply them at a later date on slave file servers. Changes should be found by comparing the status of the master and the slave file server. Comparison should be stateless – it should not depend on update history. Each file server is administrated by independent system administrator groups, so we don't want to rely on what we think the status is, but we rather have to check the actual status of the remote file server. We have to detect changes applied by remote administrators.

Update Programs Currently Executing

Files that are updated may not be overwritten. The old file has to be moved and unlinked, then the new file has to be moved to it's final destination.

Do Not Require Root Permission To Run

We install all software using unprivileged accounts and try to avoid using root permissions as much as possible. Synchronizing software file servers should be done using an unprivileged account, too. If root permission are required (e.g., to update entries in system files, or programs that require an user or group s-bit with a system owner- or group-ship), a script should be created, that is executed separately by the system administrator of the slave file server.

Support Mapping

To allow localization, a flexible mapping of, e.g., file- and path-names, permissions, and owner should be supported. Software packages might be owned by different accounts on different servers. Symbolic links replace files to implement site specific changes (e.g., for configuration files).

Support Execution of Scripts Before and After Updates

Before and after an update or roll-back it should be possible to execute scripts on the server and client. You might want to shutdown a database server and restart it after the update has finished. License servers might have to be re-started, if license files were updated.

Transfer Data Efficiently and Reliably

The data transfer must be efficient, because our links are slow and have a high latency. If the link fails for a short period of time and the data transfer is aborted, transmit only the missing data, do not re-transmit all data.

Be Humble

Do not require special installation of software packages. We don't want to change the installation of commercial software and have to support a variety of different package types.

Should Support Roll-back

It should be possible to undo at least one update. If an update of a software package introduces problems, it should be possible to go back to the previous state of the software package. Also, if an update fails, roll-back the already applied changes to return to a consistent state.

Minimize Inconsistent States

The time that a software package is in an inconsistent state (the time between the first and the last change that is applied) should be as small as possible. The time between applying changes to software packages that depend on each other should be as small as possible, too. Roll-back changes, if an update did not complete successfully.

Avoid Errors Pro-actively

Try to verify in advance, whether an update is likely to succeed, e.g. check whether the target server has enough disk space to store new or changed files.

Should Be Flexible

It should be easy to choose alternative distribution media: e.g., tape, email, direct network link. Comparing the status of master and slave file servers should be possible, even if no direct network link exists between the servers. The tool should be modular and extensible. Tool interfaces should exist and be well-documented.

Should Use Standards

Use well-known existing standards and standard tools as much as possible. Do not re-invent wheels.

Should Be Cost Effective

The cost for the product, its installation and customization, and its maintenance must be acceptable.

Evaluation of Alternatives

We took a look at freely available tools, as well as commercial tools, proposed standards, and papers dealing with software management ([14], [17], [20], [21]).

Freely Available Tools

The tools that we looked at can be categorized as follows: Tools that help to maintain source code and install software in a heterogeneous platform environment, like rtools [22]; Tools whose primary focus is network and disk space efficient installation and an unified setup and access by users in a campus network – tools like "The depot" [3], [18], depot-lite [7], opt_depot [24], ldd [4], lude [8], and beam [19] fall in this category; and tools that are designed to distribute software, like *rdist* [1], fdist [6], mirror [25], track [9], sup [2], and SPUDS [5].

All tools lack efficient incremental software distribution over slow WAN links. Bad assumptions include the set-up of software packages which imposes too tight restrictions. This won't work in our multi-vendor system environment. None cares about controlling the transmission in terms of media, scheduling, interruption or measurement and self-adoption. No roll-back support exists.

Commercial Tools

Some commercial data distribution tools exist, as well as software management tools, that provide additional functions to cover a broader range of the software life-cycle, e.g., packaging, installation and de-installation. XFer from ViaTech, MLINK/ACM & DistribuLink from Legent, Tivoli/Courier from Tivoli, and DSM-SAX from SNI fall more into the data distribution category, whereas HP OpenView Software Distributor from Hewlett-Packard, and Sun-DANS from Sun Microsystems [13] fall into the latter category.

GUI and object-oriented methods and policies ease software packaging and automate distribution and gathering tasks. Typical application fields for these commercial tools are large companies with diverse offices with many client machines like financial or insurance companies.

All commercial tools claim to comply to standards, although it is sometimes hard to tell which standard they mean. Only one commercial tool purports to be compliant to the draft of the POSIX standard 1387.2 (formerly 1003.7.2) *Software Administration*.

We found it difficult to explain exactly what we mean by *incremental update* to some tool vendors. No package had proper mechanisms built-in. Incremental updates can be added to most commercial tools by writing scripts. Price is also a problem. Truly powerful tools won't start below $50,000 just for the licenses. Add an equivalent amount for installation, customization, maintenance, and updates. One benefit of commercial tools is to reduce the required skill and cost of the personnel at remote sites. This will not work for our few, demanding development sites.

Usage of commercial tools is problematic if you want to establish links to external companies. You need to buy licenses, so has your partner, too. All evaluated distribution software tools require that on both target and source locations daemons are running, and you have to pay a license fee per master and per client.

It's foreseeable that not all companies (esp. small consulting groups) are willing or able to spend the extra money and the extra effort of installation. Therefore it's very important for us to have a tool that can be used without any restrictions at least at the client side.

Two software packages looked promising, though:

Tivoli/Courier

Tivoli Systems implemented an extensive collection of system administration tools. One of them, Tivoli/Courier, allows automatic software distribution and control of server and workstation configuration. Tivoli/Courier is embedded into the Tivoli Management Framework. Together with other tools this forms a complete management environment. A graphical user interface and an object oriented approach allow easy maintenance of large systems. Tivoli/Courier allows to define software packages, different styles of scheduling, to define which files are updated at what time. Scripts can be added to customize the management environment to special requirements.

Tivoli/Courier does not fit to our requirements in respect to incremental distribution as we need it (It could be implemented by external scripts). It was not clear if we could run Tivoli/Courier stand alone without the framework or the other system administration tools. A direct network link is mandatory. All hosts involved in the update process need licenses and the software has to be installed as root. Reference customers seem to have a different profile (many hosts to update, static package design, smaller volume) than we have.

If we already had Tivoli Management Environment in productive use as the basis of our system administration, it would make sense to evaluate the performance of Tivoli/Courier. For the time being it would be too costly to implement the Management Environment to just use the Tivoli/Courier part.

XFer

XFer from ViaTech Technology was the second tool we evaluated very closely. XFer is optimized to solve the standard software distribution task: Update many hosts spotted over the globe with packages of (in our opinion) modest size and well-known structure. Compression and packaging of updates are standard. Packages, hosts and other resources are objects and can be managed efficiently. Machines can be grouped in "profiles". These profiles allow to send updates to machines which require certain software, regardless of type and location.

But at the time of the evaluation there was no support built in for incremental distribution (in our terms). High entry costs would pay off for many client hosts, but not for the few servers we run. Database and protocol overhead are not known. XFer would profit if installed together with cooperative network, user administration and configuration management tools, which are not available on our sites.

POSIX 1387.2: Software Administration

P1387.2 provides the basis for standardized software administration. It includes commands to install, remove, configure, and verify software. A distribution format (install image) of software is defined along with commands to create and to merge distribution images. Provision is also made for tracking what software is installed and what its level is. Commands may be directed on one system to occur on any number of systems throughout your network.

There is a set of concessions to operating systems not based on POSIX.1 and POSIX.2, and there are exception conditions, so that systems such as DOS can conform to P1387.2.

Status

P1387.2 has passed its ballots within IEEE and has become the first POSIX system administration project to complete its work. Now P1387.2 has to be approved by the IEEE Standards Board, registered by ISO as a Committee Draft, and soon will be balloted as a Draft International Standard.

Copies of the current draft, P1387.2/Draft 14a, April 1995, are available from the IEEE Computer Society and from the IEEE Standards Office. A previous version of the draft (P1387.2/Draft 13) is also available by anonymous ftp [16]. See [15] and [23] for a more detailed discussion of the status of P1387.2 as of April 1995.

Suggestions for follow-on activity include a guide to best use of the current standard, a profile for DOS (and related) systems, version and patch/fix management, policies in distributed management (especially related to the definition of the "success" of an operation) and associated recovery policies, file sharing and client management, hierarchical distribution, scheduling, and queuing and queue management.

Because P1387.2 does not specify the means by which distributed functions occur, the System Management Working Group at X/Open are working to provide the necessary specifications to permit distributed interoperability.

Relevance

P1387.2 focuses on the distribution of software by installation. Software Service (patching software) has explicitly been left out of the standard, because existing schemes currently in use were too diverse. A possible solution is described in the rationale, though.

Once ISVs and all our internal developers of software, technology and library data use P1387.2 for their products, initial installation of software might become easier (or at least more similar). Hopefully even software service (i.e., applying patches) will eventually become standardized.

However, unless all our changes to all our software is done using standard procedures, we will have to clone file servers. Even if we were able to install all changes in a standard way, we would need some kind of queuing, so that we first can test the changes, before all servers are updated. Updates would have to be scheduled for night time. We don't believe that there will be a standard or a commercial product for cloning file servers any time soon.

OpenDist

No available tool solved all important requirements, so we decided to implement our own tool set. OpenDist consists of administrative tools taking care of scheduling updates, statistical tools to report changes and performance of updates, and distribution tools do the actual update. Design goals are modularity and flexibility. The tools should be independently usable entities, easily exchangeable and should work together in changeable configurations. All tools are implemented in Perl [12], Version 5. Wherever possible, existing standard tools are used: e.g., GNU tar (*gtar*(1)) *gzip*, etc.

All tools currently use a command line interface. A more convenient graphical user interface for casual users will be added using TkPerl or a HTML browser.

Administrative Tools

These manage the scheduling of updates and call the distribution tools. Software administrators can subscribe and unsubscribe to software packages, query the software package database for information about each software package, temporarily postpone the update of selected packages, force an immediate update of selected packages, or roll-back updates. There are tools to browse the update history and to retrieve information about the status of each file server and software package.

Information about our software packages is stored in a flat (ASCII) file database, and includes: name and purpose of the package, status (test, old, current), dependency between packages, grouping of packages into bundles, recommended update frequency, contact information of package maintainer, etc.

Statistical Tools

These display performance information. Transfer rates and update duration are indicators of bottlenecks and problems with WAN lines. We need early indicators to be able to upgrade our network in a timely manner. The update history can be shown, as well as the current and historic free disk status on the file servers.

Distribution Tools

These distribute software packages, replacing *rdist* in our environment. The tools are optimized for low speed links with high latency. Software updates are transmitted in a compressed format.

They try hard to never leave a package in an inconsistent state, i.e. an update must be completed or has to be rolled back. Pre- and post-processing scripts are supported to save files before updating, or restart a license server after updating.

The OpenDist Distribution Engine

The OpenDist distribution engine is implemented in several independent stages. Each stage has clearly defined input and output data formats. Tools can be easily combined and exchanged as long as the interface does not change. The different steps can be pipelined and performed in parallel when updating several software packages to further speed up the update and to optimized resource usage. For lines that do not support independent data transfers in both directions, the sending of updates and the retrieving of index files should not be done in parallel.

For our important packages we run the update once a day on the distribution server. The distribution process is controlled and mainly run on this

server. We do not use the master file server but a dedicated machine as the distribution server. This server must have a direct access to all file servers. The index files and archives are temporarily stored on a holding disk.

The Update Flow

A package is updated in the following steps:

- INDEX
  Create an index of the software package on the master and all slave servers. Compress and transfer to the distribution server's database.
- COMPARE
  Compare the master index against each slave index. Output a list of changed file attributes. Exceptions are handled here.
- BUILD-ARCHIVE
  Build a compressed archive with all changed files.
- BUILD-INSTALL
  Build an installation script which does the actual update on the slave server and takes care of changed attributes.
- BUILD-RESTORE
  Build a rollback script, which allows to undo the update in case of failure or in case the last status should be restored.
- TRANSMIT
  Transmit the archive and the scripts to the slave server.
- INSTALL
  Execute the installation script on the slave server and notify the administrator about success or failure.

The Update Stages in Detail

The INDEX Stage

Every time an update cycle is started, first a sorted index of the software package is retrieved. The index is not stored on the remote system, but immediately compressed and piped to the master server. It is plain ASCII and sorted alphabetically by filename to make comparison of indices easier. Each line in the index file describes one object of the filesystem, e.g., a file, a directory, a FIFO, etc. The *perl* script *od_getindex* is started on the master and the slave servers from the distribution server. *od_getindex* scans the complete tree beneath a given directory. Symbolic links are not traversed. For each entry the returned index contains the attributes "name", "type & permissions", "size" and "modification time" by default. For symbolic links the link target is appended. For files with a link-count greater than one, the link count and the inode number are appended. Optionally more attributes like owner, group-id or checksum can be reported. The core of *od_getindex* is very simple and efficient (Figure 1). Of course, for the final version, we added more error checks.

The attributes are separated by a tab. Files with funny characters in their name (e.g., tabs, newlines) are ignored and *od_getindex* will issue a warning message. A leading hash sign marks extra data or comment lines. Besides these characters no restrictions are imposed on filenames or package names. Usage of these and other non-printable characters should be avoided, anyway.

```
sub scan_it {
    my($Name) = @_; my(@filenames);

    ...

    ($dev, $ino, $mode, $nlink, $uid, $gid, $rdev, $size,
     $atime, $mtime, $ctime, $blksize, $blocks) = lstat($Name);

    ... if file not readable, print warning and return ...

    printf("%s\t0%o\t%d\t%d\t%d\t%d%s\n", $Name, $mode, $size, $mtime,
                $nlink, $ino, -l $Name ? "\t->\t" . readlink($Name) : "");

    if (-d $Name && (! -l $Name)) {
        opendir(DIR, $Name);
        @filenames = sort(grep(!/^\.\.?$/, readdir(DIR)));
        closedir(DIR);

        for (@filenames) {
            &scan_it(sprintf("%s/%s", $Name, $_));
            }
        }
    }
```

Figure 1: Excerpt from *od_getindex*

Extra data like a time-stamp, the remote host name, blocksize in this file system or the mapping of user id to user name are added case by case. The format of the entries is described in "#FORMAT" lines.

The output from *od_getindex* is run through an ignore filter, compressed and finally transmitted to the distribution server. The index file uses complete filenames relative to the root of the software package and gets quite large (about 100 MBytes for our 25 GBytes of software), on the other hand, the compression rate is obviously very good (about a factor of 10).

The ignore filter is an optional script, which, e.g., allows to avoid the transfer of unwanted data like huge working directories. Ignore filters can be implemented site and/or package specific. The appropriate ignore filters will be transferred from the distribution server in advance.

Other output filters (e.g., for encryption) can easily be added to the stream. At this stage there is no difference between master and slave servers. The very same program is run on all locations where the package is found.

The COMPARE Stage

As soon as the index for a master and a slave package is available, differences can be calculated. *od_compare* opens both index files and reads them line by line. As the index files are sorted by entry name, missing or extra entries are recognized easily. An `ADD` or `DEL` tag followed by name and type is printed. Following are a list of attributes and values relevant for this type.

If an entry is in both index files, all significant attributes of this entry are compared. A list of changed attributes is printed.

A post processing filter cares about well-known exceptions. These exception filters can be defined per package and/or per site. Several possible actions like "notify administrator", "ignore this" or "run script" can be triggered. An typical case is to exclude printer configuration files from being updated.

To allow for mapping of specific attributes, an optional mapping filter can be applied on the master index before comparison. Examples are mapping of user-id or names of entries.

The BUILD Stage

The remaining changes are handed over to the *od_build* script. This script reads in the changes and decides what action should be triggered, based on the type attribute. The build script packs all files, which need an update, into an archive. Currently we are using *gtar* to create the archive.

The build-update script counts the number of changes and the size of these changes. If specific built-in limits are reached different actions might be started. E.g. if the overall size of an update archive is very large, a tape might be written and sent instead of transmitting it over the direct link. If the

```
#Start "od_getindex /pool3/gnu", host: server, time-stamp: 806615319
#FORMAT_FILE     NAME     TYPM      MTIM     SIZE
#FORMAT_HARDLINK NAME     TYPM      MTIM     SIZE    NLNK    INOD
#FORMAT_SOFTLINK NAME     TYPM      MTIM     SIZE    ->      TARG
#BLOCKSIZE: 1024

... Directory:

gnu/sun4.1/bin          040775 806615225     1024

... File with linkcount > 1:

gnu/sun4.1/bin/gzcat  0100755 806614252   65536 2  75248
gnu/sun4.1/bin/gzip   0100755 806614252   65536 2  75248

... Symbolic link:

gnu/sun4.1/bin/gzcmp  0120777 806615225       6 -> gzdiff
gnu/sun4.1/bin/gzdiff 0100755 746115780    2008

... Plain file:

gnu/sun4.1/bin/gzexe  0100755 746115781    3864
gnu/sun4.1/bin/gzgrep 0100755 746115781    1341
...
#End "od_getindex /pool3/gnu", host: server, time-stamp: 806615321
```

Figure 2: Excerpt from *od_getindex* output

number of changes is above a given percentage, the administrator is notified.

Besides the archive, an installation and a roll-back script are generated. The installation script contains appropriate code for every change. It starts pre- and post-installation scripts. It checks for available disk-space on the target system and tries to avoid inconsistent states. The roll-back script is able to undo the latest update, even if the update failed half-way.

The TRANSMIT Stage

As soon as the archive, the install and the roll-back script are ready, they are scheduled for transmission. The actual transmission can run on a direct link, on tape or by email. The current implementation relies on a direct link.

The update package is transmitted in fragments suitable for the line speed. Each fragment gets a checksum and will be retransmitted in case of an error. The fragments might be encrypted. In case the package is going to be transmitted over a leased line, a flag can be set to postpone the transmission until non-busy hours, to avoid resource conflicts with other users. The fragments are decrypted and concatenated on the target system.

The INSTALL Stage

After the complete update package is on the slave server, the install script is started. At first the install script checks if the necessary disk space is available. Next all files which will be updated or deleted are written to a roll-back archive on the slave distribution holding disk.

Care is taken, that files are not overwritten. Old files are moved and unlinked, before new files are installed. This way, we can update programs that are currently executing.

Then the changes are applied. The install script runs all pre- and post-installation scripts at the right time.

If something fails with the update, the roll-back script is called to restore the latest consistent state.

Through the five independent steps changes can be implemented more easily. We could implement

```
#Start "od_compare gnu.stat.2 gnu.stat", host: od_host, time-stamp: 806616691
#MASTER #Start "od_getindex /pool3/gnu", host: server, time-stamp: 806615319
#CLIENT #Start "od_getindex /pool2/gnu", host: client, time-stamp: 806614037

... Changed attributes of file:
Change-MTIM: gnu/sun4.1/bin/gzcat     FILE  806614252 753610106
Change-PERM: gnu/sun4.1/bin/gzcat     FILE  0755      0777

... Replace a file with a symbolic link:
Change-TYPE: gnu/sun4.1/bin/gzcmp     SYML  SYML      FILE
Change-SIZE: gnu/sun4.1/bin/gzcmp     SYML  6         2008
Change-PERM: gnu/sun4.1/bin/gzcmp     SYML  0777      0755
Change-MTIM: gnu/sun4.1/bin/gzcmp     SYML  806615225 746115780
Change-TARG: gnu/sun4.1/bin/gzcmp     SYML  gzdiff    <UK>

... Added a symbolic link:
ADD:         gnu/sun4.1/bin/zcat      SYML  5
Change-PERM: gnu/sun4.1/bin/zcat      SYML  0777      <UK>
Change-TARG: gnu/sun4.1/bin/zcat      SYML  gzcat     <UK>
Change-MTIM: gnu/sun4.1/bin/zcat      SYML  806614148 <UK>

... Deleted a directory + files in it:
DEL:         gnu/sun5.2/lib           DIR   1024
Change-PERM: gnu/sun5.2/lib           DIR   <UK>      0775
Change-MTIM: gnu/sun5.2/lib           DIR   <UK>      806612281
DEL:         gnu/sun5.2/lib/libfl.a   FILE  1328
Change-PERM: gnu/sun5.2/lib/libfl.a   FILE  <UK>      0644
Change-MTIM: gnu/sun5.2/lib/libfl.a   FILE  <UK>      801569606

... Change the target of a symbolic link:
Change-SIZE: gnu/sun5.4/man           SYML  16      11
Change-TARG: gnu/sun5.4/man           SYML  ./share/man/man3 ./share/man
#End "od_compare gnu.stat.2 gnu.stat", host: od_host, time-stamp: 806616694
```

Figure 3: Excerpt from compare output

different INSTALL backends and use the most appropriate case by case.

The separation of finding and applying changes gives us more flexibility in trying not to run into disk space problems: We could remove all files that are going to be changed at the beginning of the update, or we could delete and add files simultaneously. We could even try to allocate extra disk space to prevent someone else from "stealing" disk space while we are installing the changes. However as long as several changes are applied simultaneously, it is not possible to guarantee that we will have enough disk space to install the changes, nor that we can roll-back to the previous status, though this is unlikely to happen.

Results

OpenDist performs much faster than our previous, *rdist* based solution. Comparing a subset of 10 % of all software on the software servers in Villach and München can be completed in half an hour instead of 8 hours.

Creating the archive of files to be distributed must be done carefully. There is no backup or archive program that can cope with every special situation (files with holes, excessive pathname or symbolic link length, unreadable files and directories, or special (device) files) [10]. Currently we are using *gtar* and will document known limitations.

Distributing an archive of files helps to guarantee a consistent state of the target server. We do not start to change the slave server before the complete set of changed files has been transmitted to the slave server's holding disk. During the actual change of the slave server's software package, the network link might be down without affecting the update process.

Roll-backs of (at least one) update can be more easily supported than with *rdist*. The installation script can archive all files to be deleted or changed. The distribution engine creates a script that replaces the changed files with the saved ones.

A link (currently a direct network link) between master and slave server is needed only in stages INDEX and TRANSMIT. The COMPARE and the BUILD stage perform locally on the distribution server, independent of the line to the remote host.

Now instead of line speed and latency, index creation and compression became the bottleneck: the transfer rate from our Singapore site would allow to transfer the complete compressed index (5.5 MBytes) in about 25 minutes, whereas it takes about three hours now.

When you access the software over NFS, index creation slows down by a factor of 3 - 5. Therefore index creation should always be done on the file server that actually stores the software.

Small changes are very likely to be bug-fixes to software officially released and in use. Therefore these have to be found and applied as soon as possible. If a package is changed substantially, it's very likely to be new, not yet officially released. Changes to such a package are not as urgent and can be postponed to weekends and non-busy hours. OpenDist is very efficient in finding differences and there is enough bandwidth available to apply small changes on the fly. This allows us to run OpenDist once every day on all packages to apply small changes and postpone major changes to weekends.

The index files can be stored and the same tools can be used not only to find differences between different sites, but also to find changes to software packages in a specific period of time. Thus the same tools can be used to track changes applied to our software packages over time.

OpenDist needs no special installation on the remote systems. We can run the OpenDist procedure with any external company without forcing them to buy licenses or to install a huge tool package. All the tools used in OpenDist are freely available and can be distributed. Thus we could deliver, e.g., perl and gnu-software to the client, too.

OpenDist uses *rsh*(1) and *rcp*(1) which requires trusted hosts. We would not use this over public networks. However within our corporate network this is acceptable. It is also possible to replace *rsh* usage by a more secure mechanism, if required.

Conclusion

Is OpenDist Better Then rdist?

Are apples better than oranges? It depends.

The OpenDist toolset has more functionality than a distribution engine like *rdist*; in fact, we could use *rdist* as one of OpenDist's distribution engines. OpenDist fulfills all our important requirements and compared to our previous *rdist*-based solution, wins especially in the categories: efficient transfer over low speed, high latency lines; support to roll-back changes; and time period in which software packages are in an inconsistent state. Initial installation and configuration needs more effort, though. You will also need additional disk space for the holding disk on all file servers.

rdist is easier to setup and configure and is a standard part of most Unix systems (if your vendor ships an old version of *rdist*, upgrade to *rdist* Version 6 and complain at you vendor about the old version.) We still use *rdist* for many smaller tasks in the local network as well as to copy few files to remote servers.

Both tools have their limitations, and OpenDist depends on the tool used to create archives, which has its own set of limitations [10].

Though we started to just replace *rdist* with a better distribution tool, we realized that software distribution is closely coupled with other software

maintenance tasks, like installing software, keeping information about installed software, de-installation, making software accessible by users. All stages of the software life cycle interact with each other. Changes made in one stage can help to solve problems in related stages.

Future Work

The first implementation of the OpenDist distribution engine has proven to be superior to *rdist* in our environment, so we will further enhance it. Enhancements include reduction of re-transmitted bytes in case of line failures, optional transfer media (e.g., use tapes to transmit large low priority packages). We will also add more functions to administrative and statistics tools. Scheduling of updates and tools to query the status of slave software servers will be implemented next.

We will fine tune the parallelization of the update to further speed up the update. We plan to implement some of the needed update functions (move file before updating) within *gtar*, which will make the installation script simpler and more robust.

Availability

At the time of writing, OpenDist is only available for use within Siemens AG. We hope to make the source for OpenDist available on the Internet at a later time. The paper and the slides for this talk are available by anonymous ftp from `ftp.ConnectDE.NET` in the directory /pub /sysadmin/sw-distribution/OpenDist/.

Acknowledgments

Special thank goes to Tom Christiansen for reviewing early drafts of this paper, to all our system administrators for valuable discussions and encouragement, to Gernot Babin for his many contributions, and to SAM for all the support you have given. We thank the companies Interchip and Opis for their support during the evaluation of their products. We also thank Connect! for providing ftp and web space.

Author Information

Peter W. Osel received his diploma in electrical engineering from the Technische Universität München (TUM) in 1985. For three years he worked at central research and development of Siemens AG, where he developed tools for ECAD of Integrated Circuits. Since 1988 he is working for the Semiconductor Division of Siemens. He is responsible for worldwide integration and distribution of the CAD system, as well as the development of central tools, and the coordination of the development sites' system environment. Reach Peter at Siemens AG, HL CAD, Postfach 801709, D-81617 München, Germany; or by e-mail at pwo@HL.Siemens.DE, or see his Web page at URL: http://www.ConnectDE.NET/people/pwo/.

Wilfried Gänsheimer received his diploma in electrical engineering from the Technische Universität München (TUM) in 1990. Since then he is working for the Semiconductor Division of Siemens. He started at the application support group for RISC microprocessors and Siemens microcontrollers. There he was responsible for definition, specification and test of development tools, customer support, benchmarking and simulation of system performance. He ran a small heterogeneous network (PC, X-Terminals, Workstations) inside the Siemens network. Since end of 1994 he is working in the CAD department. Management of licenses, installation of software, software distribution and user support are the main topics. Trouble shooting printer/plotter problems is his favorite time waster. Reach Wilfried at Siemens AG, HL CAD, Postfach 801709, D-81617 München, Germany; or by e-mail at wig@HL.Siemens.DE.

References

[1] Michael A. Cooper, "Overhauling Rdist for the '90s", *Proceedings of the Sixth Systems Administration Conference (LISA VI)*, pp.175-188, Long Beach, CA, October 19-23, 1992

[2] Stephen Shafer and Mary Thompson, "The SUP Software Upgrade Protocol", Carnegie Mellon University, School of Computer Science, 1988. Available from mach.cs.cmu.edu in /usr/mach/public/doc/sup.ps.

[3] Wallace Colyer and Walter Wong, "Depot: A Tool for Managing Software Environments", *Proceedings of the Sixth Systems Administration Conference (LISA VI)*, pp.153-162, Long Beach, CA, October 19-23, 1992

[4] Walter C. Wong, "Local Disk Depot – Customizing the Software Environment", *Proceedings of the Seventh Systems Administration Conference (LISA VII)*, pp.51-55, Monterey, CA, November 1-5, 1993

[5] Ola Ladipo, "A Subscription-Oriented Software Package Update Distribution System (SPUDS)", *Proceedings of the Workshop on Large Installation Systems Administration*, pp.75-77, Monterey, CA, November 17-18, 1988

[6] Bjorn Satdeva and Paul M. Moriarty, "Fdist: A Domain Based File Distribution System for a Heterogeneous Environment", *Proceedings of the Fifth Large Installation Systems Administration Conference (LISA V)*, pp.109-125, San Diego, CA, September 30 - October 3, 1991

[7] John P. Rouillard and Richard B. Martin, "Depot-Lite: A Mechanism for Managing Software", *Proceedings of the Eighth Systems Administration Conference (LISA VIII)*, pp.83-91, San Diego, CA, September 19-23, 1994

[8] Michel Dagenais, Stéphane Boucher, Robert Gérin-Lajoie, Pierre Laplante, and Pierre Mailhot, "LUDE: A Distributed Software Library", *Proceedings of the Seventh Systems Administration Conference (LISA VII)*, pp.25-32, Monterey, CA, November 1-5, 1993

[9] The track package is available at URL: ftp://ftp.cs.toronto.edu/pub/track.tar.Z

[10] Elizabeth D. Zwicky, "Torture-testing Backup and Archive Programs: Things You Ought to Know But Probably Would Rather Not", *Proceedings of the Fifth Large Installation Systems Administration Conference (LISA V)*, pp.181-189, San Diego, CA, September 30 - October 3, 1991

[12] Larry Wall and Randal L. Schwartz, *Programming perl*, O'Reilly & Associates, Inc., Sebastopol, CA, 1991

[13] SunDANS (SoftDist) Manual, alpha draft, Sun Microsystems Computer Corporation, Mountain View, CA, September 1993

[14] Ram R. Vangala, Michael J. Cripps, and Raj G. Varadarajan, "Software Distribution and management in a Networked Environment", *Proceedings of the Sixth Systems Administration Conference (LISA VI)*, pp.163-170, Long Beach, CA, October 19-23, 1992

[15] Barrie Archer, "Towards a POSIX Standard for Software Administration", *Proceedings of the Seventh Systems Administration Conference (LISA VII)*, pp.67-79, Monterey, CA, November 1-5, 1993

[16] "POSIX 1387 System Administration Standard (draft 13)", available at URL: ftp://dcdmjw.fnal.gov/posix/

[17] John Sellens, "Software Maintenance in a Campus Environment: The Xhier Approach", *Proceedings of the Fifth Large Installation Systems Administration Conference (LISA V)*, pp.21-28, San Diego, CA, September 30 - October 3, 1991

[18] Kenneth Manheimer, Barry A. Warsaw, Stephen N. Clark, and Walter Rowe, "The *Depot*: A Framework for Sharing Software Installation Acros Organizational and UNIX Platform Boundaries", *Proceedings of the Fourth Large Installation System Administrator's Conference*, pp.37-46, Colorado Springs, CO, October 18-19, 1990

[19] Thomas Eirich, "Beam: A Tool for Flexible Software Update", *Proceedings of the Eighth Systems Administration Conference (LISA VIII)*, pp.75-82, San Diego, CA, September 19-23, 1994

[20] D. Nachbar, "When Nework File Systems Aren't Enough: Automatic Software Distribution Revisited", USENIX Conference Proceedings, Summer 1986, pp.159-171

[21] Steven W. Lodin, "The Corporate Software Bank", *Proceedings of the Seventh Systems Administration Conference (LISA VII)*, pp.33-42, Monterey, CA, November 1-5, 1993

[22] Helen E. Harrison, Stephen P. Schaefer, terry S. Yoo, "Rtools: Tools for Software management in a Distributed Computing Environment", Proceedings of the Summer 1988 USENIX Conference, pp.85-94, San Fancisco, CA, 1988

[23] Nicholas M. Stoughton <nick@usenix.org>, "Standards Update, POSIX System Administration: Software Administration", Newsgroups: comp.std.unix, Message-ID: <3rsf12$nqu@cygnus.com>, 16 Jun 1995 10:29:06 -0700, available at URL: ftp://ftp.ConnectDE.NET/pub/sysadmin/sw-distribution/OpenDist/P1387.2-status-9504

[24] URL: http://www.arlut.utexas.edu/opt_depot/opt_depot.html

[25] URL: ftp://src.doc.ic.ac.uk/packages/mirror/

filetsf: A File Transfer System Based on lpr/lpd

John Sellens – University of Waterloo

ABSTRACT

In a distributed computing environment, it is often necessary to transfer files between machines for administrative or record keeping purposes. Most methods of file transfer either require full-scale trust by the recipient (e.g., `rdist`, `rcp`), the use of a "secret" password (e.g., FTP), or some level of pre-arrangement (e.g., `rdist`, `track`). This paper describes `filetsf`, a generic file transfer system built on top of the standard `lpr/lpd` system. `Filetsf`'s advantages include minimal configuration, the need for minimal trust between sender and recipient, ordering and spooling of file transfer requests, and the elimination of the need for shared passwords.

Introduction

The University of Waterloo is in the process of converting its administrative systems from legacy implementations on IBM's VS/1 operating system under VM/CMS and the PICK database system on UNIX, to implementations based on the Oracle RDBMS on UNIX database and application servers. As part of the ongoing conversion, we have a need for regular (and irregular) file transfers between machines, primarily from the VS/1 and PICK systems to the Oracle systems, but also between UNIX systems.

To date, we have used a number of approaches, each of which seems to have had drawbacks, primarily security and reliability related. We needed a system to allow file transfers between our systems that allowed us to limit each programmer's sphere of control, i.e., we wanted to avoid giving fullscale access to all systems and servers, just to enable file transfers. We also needed a file transfer system that was reasonably portable, and worked on UNIX and non-UNIX operating systems.

The end result was the `filetsf` family of programs, that provides the ability to send and receive files between cooperating machines, is implemented on top of the Berkeley `lpr/lpd`[3] system, and which provides a suitable level of flexibility while retaining access controls and an acceptable level of security.

Previous Approaches

In the past, we have primarily used FTP[11] for file transfer – this meant that most programmers and operators knew the passwords to any accounts that needed to receive files. This was not a good thing, since shared passwords are no longer secret, and because it allowed general use access to accounts that shouldn't have been used for anything except file transfer. And, in the absence of digital signatures, FTP does not allow the recipient to identify the sender once a file has been delivered, making it potentially easy for legitimate data to get replaced, intentionally or not.

We implemented a simple FTP-based system using a single recipient userid (named 'unifer', for UNIX transfer), where all files sent to a machine would end up in a single directory, in a particular group, with group write permission on everything. This meant, of course, that any file transferred to a machine was fair game for anyone else to access. Most file transfers were reasonably well-controlled, but left-over, unclaimed, files tended to accumulate, and it's hard to make a claim that this system provided a reasonable level of security and auditability.

We also used `rdist(1)`[1] in a few instances, which either required setup and monitoring by the (limited number of) super-users, or which required '.rhosts' file access to the receiving account, with many of the same problems as with FTP. `rdist` also isn't available on our VM/CMS system, which meant that it could not be used for the bulk of our file transfers. `rcp` suffers from many of the same problems as `rdist`, while providing a less-reliable copy to the recipient machine (`rdist` uses temporary files to ensure that the destination file is always complete, `rcp` just clobbers any existing file, and can leave files incomplete if interrupted or the destination disk fills up).

We haven't investigated `track`[9], because we have no experience with it on campus, and because, like `rdist`, it requires a level of pre-arrangement that we wanted to avoid.[2] `track`, like `rdist`, is also geared towards the replication of a set of files on multiple machines – for file transfer, we need to be able to send a file and then remove the original.

[1] An enhanced version of `rdist` is described in [5].

[2] We may be investigating `track` for use in software package distribution, under our **xhier**[12] software maintenance system.

None of these approaches allows for the queueing of file transfer requests if the recipient or network is down, although it is possible to periodically repeat the `rdist` or `track` commands. And they don't provide for an ordering of the files transferred, unless you name the files with indicative names, or use file modification times as an indicator. In our previous systems, if a series of files needed to be processed (e.g., batches of transactions), the files were typically sent one at a time, processed on the recipient machine, and an acknowledgement was returned to the sender (usually via a flag file) to indicate that it was safe to send the next file.

Desired Features

We identified the following features as desirable in a file transfer system. Many of these are convenience features, intended to help enable a more automated environment, with less manual intervention (e.g., through avoiding entering passwords, or having to manually re-initiate transfers in case of failure).

- Available on all or most of our systems.
- Requires minimal configuration – we wanted and needed to be able to do ad-hoc file transfer.
- Allows file transfers between random pairs of users – root-to-root file transfers would not be enough, since we need to allow our programmers to use the file transfer system, none of whom are super-users.
- Allows the recipient to identify the sender and name of the file – a recipient could be receiving multiple copies of identically named files from more than one sender. For example, similar systems on different machines might generate transaction files to feed to a central server.
- Allows an ordering of files – e.g., files of transactions should be receivable in the order in which they were sent.
- Provides for queueing of file transfer requests, in case of machine or network unavailability. We didn't want to have to keep retrying file transfers manually if the destination machine was down or unavailable.
- Provides for a non-clashing name space – we wanted to be able to send multiple files with the same name, distinguished by sender, recipient, and/or time.
- Does not allow any additional access to the sending or receiving machine. We can't allow users, applications, operators or programmers to have access to anything more than they need to have access to.
- Provides a reasonable level of security in our environment. For most of our transfers, elaborate security is not required; we have a reasonable level of confidence in the security of our systems and networks. Encryption and digital signatures were not required in the basic file transfer system.
- Is based on file transfer, not file replication. We need to be able to generate a file, send it, and then destroy the original.
- Can operate unattended. Too many of our current systems require manual intervention from an operator, and there is increasing pressure (fiscal and otherwise) to avoid the need for manual processes in our systems.

VM/CMS has the `sendfile` command, which allows file transfer between users on VM/CMS systems – the file shows up in the recipient's "reader", tagged with the sender, filename, and file attributes. It meets many of our requirements, but we could not use it – we had no way to send the files to our UNIX machines from our VM/CMS machine, and we didn't have a VM/CMS compatible `sendfile` command (or equivalent) on our UNIX machines.

Approaches Considered

We were not aware of any publically available file transfer systems, and were trying to avoid investing in (and investigating) commercial solutions that might exist. We felt that our best approach would be to implement our own file transfer system.

Past LISA proceedings have contained quite a few papers on software distribution (xhier[12], depot[7], lude[6], lfu[1], CMU depot[4, 13], etc.), typically based on the idea of software "packages" that are replicated from master machines to slave machines, typically using `rdist`, NFS mounts, or FTP. These systems don't meet our needs, since they are based on the idea of replication of predefined sets, not the transfer of individual files. The tools used to transfer files (`rdist` and FTP) are not suited for our needs (as described above), and we were unwilling to investigate the use of NFS mounts, since that opens up a whole different can of worms, and since NFS is not available on our VM/CMS system.

Early in 1993, we started planning the construction of a complete file transfer system, with authentication, queueing, and so on, but we didn't make much headway, for a variety of reasons. That project never received much attention, and was eventually abandoned, and was set aside until a better opportunity presented itself.

It often seems to be the case that a problem left to its own devices eventually presents its own solution, and we finally realized, after much time had passed, that it would be relatively simple to implement a file transfer system on top of `lpr/lpd`. The `lpr/lpd` system has much of what we required, and it turned out to be relatively easy to layer the additional functions on top of `lpr/lpd` to obtain a file transfer system that met our needs.

Why not implement a file transfer system on top of SMTP[10] mail and MIME[2] file encapsulation? Mail doesn't meet some of our needs: it's too easy to forge, and it doesn't provide a reliable ordering of the messages. Digital signatures (such as those provided by PGP[14]) could be used, but they add another level of complexity, and may be harder to port to our non-UNIX systems. In addition, the path a mail message takes can't always be controlled (in the presence of MX records in the domain name server), and some mailers implement limits on the size of a message.

Why Use lpr/lpd?

It turned out that `lpr/lpd` and the LPD protocol[8] were a pretty good basis for a file transfer system. The protocol is well-defined, and widely implemented, and `lpr/lpd` already has queuing, file ordering, sender identification, machine-based access control, and so on built-in. The destination machine can be indicated by the "printer" name used, an `lpr/lpd` "output filter" can be used to process the files at the destination machine, and the job and class options to `lpr` can be used to identify the name of the file, and the intended recipient.

Using `lpr/lpd` was appealing because it seemed that it would save us a lot of effort – most of the hard stuff was already done – and because we could claim that we were behaving in a manner that was consistent with the "UNIX philosophy": small, simple, reusable tools put together to do more interesting things.

It was fairly easy to implement `filetsf` on top of `lpr/lpd`. The `sendfile` command is just a cover for `lpr`, `tsfif` is an `lpr/lpd` output filter that deposits the files into the `filetsf` spool directory, and we just have to set up a "sending" print queue for each machine we want to send to, and a a single "receiving" queue on each "destination" machine. In our case, the number of machines that we want to be able to send files to is manageably small, and our 'printcap' file maintenance tools help a lot. An `lpr` client program[3] could be used to reduce the number of "sending" queues required, but this would mean that there would be no sender-side queueing, which is one of the features that we were interested in. One non-trivial program is required – `acceptfile` is a setgid program used to query and retrieve files from the `filetsf` spool directory.

One final indication that `lpr/lpd` and the LPD protocol were a good choice was the relative ease with which we could implement `filetsf` on our AIX systems (on top of IBM's queueing system, which is willing to use the LPD protocol to talk to other machines) and implement the sending side on our VM/CMS machine with a REXX script on top of the `netprint` command. We haven't investigated implementing the recipient side on VM/CMS yet, but we're optimistic that it shouldn't be too difficult, perhaps requiring just another REXX script running in a virtual machine that invokes the VM/CMS `sendfile` command.

Implementation Details

`Filetsf` works by accepting files to send to a recipient, submitting them to `lpr/lpd`, delivering them into a spool directory, and waiting for the intended recipient to come and pick them up. A separate `filetsf` spool directory is used for easier access and control, and to provide a level of abstraction above that of particular `lpr/lpd` implementations. It also makes permissions easier to control, and keeps our monitoring software from complaining that the print queues are stalled.

Files are stored in the `filetsf` spool directory in a hierarchy intended to make it easier to list and identify the files waiting to be retrieved. The structure is three levels deep below the spool directory, encoding the recipient, intended filename, and sender (user@host) in the directory names. The individual copies of files are named for the value of `time()` (the number of seconds since January 1, 1970) when the file was delivered to the spool directory by `tsfif`, with an optional prefix. This means that the pathnames under the spool directory are of the form 'recipient/name/sender/[prefix]time'. This structure allows multiple copies of the same file to be spooled from one or more senders for one or more recipients, providing a different pathname for each spooled file. Files are named for the time at which they were delivered using the integer returned by `time()` to allow for easy manipulation and sorting. Any name clashes discovered by `tsfif` can be resolved by waiting one second and re-generating the pathname.

Most special characters are prohibited in file, sender and recipient names, to avoid potential problems manipulating the files. We had considered making the spool hierarchy only one level deep, but we felt that that would have forced even more restrictions on the component names – we would have had to use some character or sequence of characters as a separator in the directory names.

A small number (currently three) of filename prefixes are used by `filetsf` to indicate the status of a particular version of a file. Temporary files, indicated by the '#' prefix, are used when writing a file into the spool directory, and are renamed to remove the prefix when they have been written correctly. A prefix of '-' is used to indicate a file that has been retrieved or deleted, but not yet purged from the system. And a prefix of '+' is used is to indicate that the newest version of the file should

[3] One `lpr` client program is available from Keith Moore of the University of Tennessee as ftp://cs.utk.edu/pub/moore/port-lpr.tar.

replace any older versions of the file still waiting to be retrieved. This is determined by the options used by the sender when initially sending the file, and can be useful for data such as a 'hosts' file, or a telephone list, where only the most recent copy of the file is useful.

Files are transferred using `lpr/lpd` one file at a time, to keep things simple. The name (or desired name) of the file is sent as the LPD job name, and the LPD class is used to pass `filetsf` options and the name of the recipient. The options are encoded as an integer, and are separated from the recipient name by a colon.[4] The LPD protocol defines limits on the length of the control values that can be passed to the destination machine, but they haven't turned out to be unduly restrictive in practice.

`Filetsf` currently supports three options that are passed to the destination machine: notify the sender by email on delivery to the spool directory, notify the recipient on delivery, and mark the file as replacing any older versions of the file.

Command Descriptions

The core of `filetsf` consists of only three commands, `sendfile`, to send a file, `tsfif`, which is used as the `lpr/lpd` print filter and which puts files into the `filetsf` spool area, and `acceptfile`, to list or retrieve files from the spool area. In addition to the core commands, there are two utility programs, and a `tsfif` cover for use on AIX. The commands are written in C and the Bourne shell, and are about 2,000 lines of code in total.

The following is a short description of each `filetsf` command.

sendfile Queues files for sending to a user on a remote machine. It provides options to provide a different name for the delivered file, to cause mail to be sent to the sender or the recipient when the file arrives, or to flag the file as replacing any previous copy of the file still in the spool directory. One the command line and files have been checked, `sendfile` invokes `lpr` on each file in turn to send the files to the destination machine.

acceptfile Retrieves or lists files from the spool directory. It provides options to select spooled files by various attributes (sender, filename, newest, oldest, etc.), to rename a file on retrieval, and to delete files from the spool directory. `acceptfile` is setgid to the 'filetsf' group, which allows it to manipulate the files in the spool directory, and is the only privileged program in `filetsf`.

tsfif Print filter that writes transferred files into the spool directory. `tsfif` is invoked by the print system once for each file, and creates the appropriate directory structure and writes the file into the spool directory, taking care that the permissions and group of the files and directories are set as correctly as possible. Temporary files are used to insure that only complete and correct files are left in the spool directory for retrieval.

`tsfif` usually ends up being run by 'daemon' ('lpd' on AIX), and the spool directory is owned by 'daemon' (or 'lpd') and is in group 'filetsf'. If BSD-style group inheritance is in not in use, 'daemon' (or 'lpd') must be added to the 'filetsf' group. If the group of the files and directories is or can be set to 'filetsf', then `tsfif` sets them group readable and writable; otherwise it leaves the permissions conservatively set, on the expectation that `ftcheck` will complain to a human about the configuration.

aixbe Print system backend program for use on AIX. The AIX printing system is IBM specific, and uses backend programs instead of print filters. `aixbe` does the appropriate things for AIX, and then calls `tsfif` to do the actual processing of a received file.

ftcleanup Cleans up the spool directory. Removes old temporary and deleted (but not purged) files, empty directories, and looks for old files that haven't been retrieved by the recipient yet and complains about them. Run daily by `cron` (8).

ftcheck Does a sanity check on the `filetsf` system. Checks for the 'filetsf' group, checks the membership of the group, and makes sure that the spool directory and the files in it have the correct group.[5] It should really check the modes of the spool files and directories, but it doesn't (yet). Run daily by `cron` (8).

The VM/CMS client implementation was fairly simple; it's a REXX script called `filetsf` (since there's already a `sendfile` command on VM/CMS) that invokes the `netprint` command to send the file to the destination machine. Our VM/CMS system also has a command called `lpr`, but `netprint` serves the same function and is easier to use. Both `lpr` and `netprint` seem to be client-only implementations – they don't provide any spooling if the remote machine is down, which means that we have to manually retry file transfers from VM/CMS if the network or the destination system is down. We have not yet determined the best solution to this problem.

[4] We had hoped to use the title option to `lpr` for the recipient, and use class just for `filetsf` options, but we found that the title wasn't included in the control file passed to the remote machine, so that idea fell through.

[5] `ftcheck` is interesting because it contains a one line `awk` program that happens to trigger a bug in `awk` on our copy of DEC OSF/1 V3.2A that causes `ftcheck` to complain incorrectly.

We hope that it will be possible to implement a `filetsf` server on VM/CMS, but have not yet investigated what it would entail. We're hoping that it will turn out to be a another simple REXX script and a little bit of configuration.

Installation and Configuration

Installation and configuration of `filetsf` is fairly straightforward. The programs require very little configuration to compile, and contain almost no machine-specific code, or non-standard library routines. The primary exception to this is, of course, `aixbe`. The programs can be installed just about anywhere in the filesystem.

There should be a sending queue named 'tsf-machine' for each host 'machine' to which files will be sent. Each destination machine needs a queue called 'filetsf' with print filter `tsfif`. Figure 1 shows sample entries. The `filetsf-config`(7) man page gives details on userids, groups, 'printcap' file entries, and queue configuration for both `lpr/lpd` based systems and AIX.

Examples of Use

`Filetsf` is intended to be easy to use. This is an very important feature for us, since we wish to convince the application programmers in our department to use this new file transfer system, rather than old familiar FTP. It's hard to convince anyone to use something that's more complicated than what they're already used to.

The simplest example of sending a file is

```
% sendfile -h otherhost file
```

which sends 'file' to the same user on host 'otherhost'.

```
tsf-machine:\
    :mx#0:\
    :rp=filetsf:\
    :rm=machine:\
    :fx=f:\
    :lf=/var/lpr/tsf-machine/log:\
    :sd=/var/lpr/tsf-machine:
filetsf:\
    :lp=/dev/null:\
    :af=/dev/null:\
    :sd=/var/lpr/filetsf:\
    :lf=/var/lpr/filetsf/log:\
    :if=/local/filetsf/etc/tsfif:\
    :fx=f:
```

Figure 1: Sample 'printcap' file entries

To distribute a file like '/etc/hosts' to a different machine, the command would be

```
% sendfile -h otherhost -r \
          -u root /etc/hosts
```

The '-r' option indicates that this copy should replace any copy still in the `filetsf` spool directory on 'otherhost', and the '-u' option indicates that the file should be sent to 'root', regardless of who is sending the file (presumably 'root' is the only user with permission to update the '/etc/hosts' file). The `sendfile` command only allows one '-h' option and one '-u' option, so multiple invocations of `sendfile` are required to send a file to more than one recipient. Multiple files can be sent by one `sendfile` command, so, for example, the '/etc/networks' file could be sent at the same time as the '/etc/hosts' file.

```
% acceptfile -l
jms
    hosts
        jms@mach1.uwaterloo.ca
            805922115 Sun Jul 16 15:15:15 1995
            805926809 Sun Jul 16 16:33:29 1995
        jms@mach2.uwaterloo.ca
           r805922175 Sun Jul 16 15:16:15 1995
    networks
        jms@mach2.uwaterloo.ca
           r805922175 Sun Jul 16 15:16:15 1995
           r805923542 Sun Jul 16 15:39:02 1995
% acceptfile -l -c
jms hosts jms@mach1.uwaterloo.ca  805922115 Sun Jul 16 15:15:15 1995
jms hosts jms@mach1.uwaterloo.ca  805926809 Sun Jul 16 16:33:29 1995
jms hosts jms@mach2.uwaterloo.ca r805922175 Sun Jul 16 15:16:15 1995
jms networks jms@mach2.uwaterloo.ca r805922175 Sun Jul 16 15:16:15 1995
jms networks jms@mach2.uwaterloo.ca r805923542 Sun Jul 16 15:39:02 1995
```

Figure 2: Sample `acceptfile` output

The `acceptfile` command is a little more complicated to use, since it may be necessary to select a particular version of a file. The '-l' option to `acceptfile` generates a list of the queued files, such as that shown in Figure 2. The output is the recipient userid, file name, sender and the instance of the file, with any repeated information suppressed. The 'r' preceding three of the file instances in the example indicates that those files were sent with the '-r' option to `sendfile`. The '-c' option to `acceptfile` can be used with '-l' to list the files in columns, for easier parsing by other programs.

To retrieve a file, `acceptfile` must be invoked with options that result in the selection of exactly one file. The selection options can also be used with '-l' in order to test the selection options. For example,

```
% acceptfile -n hosts
```

would retrieve the newest version of the file 'hosts' into the current directory. The '-o' option can be used to select the oldest version of a file, so that files can be processed in the order they were sent. A file labeled with a particular time can be selected with the '-t' option, as in

```
% acceptfile -t 805922175 hosts
```

Files can be selected by sender with the '-s' option, as in

```
% acceptfile -s \
  jms@mach2.uwaterloo.ca hosts
```

The retrieved file can be renamed or placed in a different directory with the '-r' option. Files can be deleted from the spool directory without being retrieved by using the '-d' option to `acceptfile`. When a file is retrieved or deleted, `acceptfile` renames the spooled copy of the file for later removal by `ftcleanup`. The '-p' (purge) option can be used to cause the files to be removed from the spool directory immediately.

Security and Reliability

As with any other business system, a file transfer system needs to be reviewed to ensure that it provides an appropriate level of security and reliability for the applications in which it will be used. We will discuss the security and reliability aspects of `filetsf` in a step by step review of the file transfer process.

In our environment, there are no new security implications in the act of sending a file. `sendfile` is not privileged, so the sender of a file must be able to read the file already. Since email and FTP are available on our systems, `sendfile` does not provide any confidentiality exposures that don't already exist. In the interest of reliability, `sendfile` does not send any files until it has tested all the files for existence and readability, so that the sending of a set of files is less likely to fail part way through.

The `lpr/lpd` system is reasonably secure and reasonably reliable – we have a reasonable expectation that we can control which machines have access to our print queues and the data flowing through them. Our administrative systems are on subnets separate from the rest of the campus, our routers can block LPD protocol traffic (if necessary), and we use a locally modified `lpr/lpd` (a precursor to PLP[6]) that allows us to implement per-queue access controls, so that we can limit the machines that we are willing to trust to exchange files with. This allows us to trust the information in an LPD control file once it reaches the destination machine, so that we can label a file as being sent by a particular sender with a reasonable level of confidence. And `lpr/lpd` tries to deal correctly with full print spool directories and network interruptions, which allows us to believe that a file that has been passed to `lpr/lpd` will eventually show up in the right place, without being corrupted along the way.

`tsfif` is run on the destination machine by `lpd`, usually as the 'daemon' user, and creates subdirectories in the spool directory and writes the transferred files into the appropriate directory. The primary concern with `tsfif` is that it needs to be careful that it does not deposit a file in the spool directory over an existing file or symbolic link. This risk is minimal, since the holding directory has no general permissions on it, making it unlikely that someone could create a dangling symbolic link in the spool directory. If `tsfif` runs into any problems, such as permission or ownership errors, or a full spool directory, it either writes the file with restrictive permissions, or fails with an exit code that causes `lpd` to keep trying to deliver the file. This means that, for some kinds of errors, the 'filetsf' print queue will stop delivering files. We chose to keep retrying because the only alternative was to give up on a file, and have that particular transfer get lost, and we felt it was very important to protect the files being transferred.

`acceptfile` is a privileged program; it is setgid to the 'filetsf' group. It is very careful about what it does since it has privileges that an ordinary user does not have, and it allows a user to make use of those privileges. `acceptfile` is very careful about identifying its invoker, and it renounces its privileges before writing a file to its final destination, so that a user should not be able to change a file with `acceptfile` that can't already be changed by the user. For reliability, `acceptfile` renames files in the spool directory, rather than

[6]The most recent version of PLP is available from ftp.iona.ie in the 'pub/plp' directory. See also "LPRng – An Enhanced Printer Spooler System" by Patrick Powell, elsewhere in these proceedings.

deleting them after delivery. If something goes wrong in later processing, and the delivered file is lost or destroyed, there may be a copy that could be recovered from the `filetsf` system (by a superuser).

The `filetsf` system, through the use of the 'filetsf' group, tries to limit its potential exposures. If the 'filetsf' group is somehow compromised, there should be limited exposure to the rest of the machine, since nothing else should be using the 'filetsf' group.

`Filetsf` isn't 100% secure or 100% reliable; as in any system, the risks have to be weighed against the potential consequences and the costs to eliminate the risks. The `filetsf` system provides, for us, an acceptable level of security and reliability.

Potential Extensions

`Filetsf` was planned to be a simple file transfer system, providing the necessary, basic functions, and providing appropriate building blocks on which to implement more enhanced functionality (we'll claim that this is the "UNIX philosophy" again). While `filetsf` seems to work just as it is, there are a number of potential extensions to `filetsf` that may be worth investigating.

Some file transfers will likely require greater security than that provided by `filetsf`, due to greater potential exposures. Since `filetsf` is just sending files, file encryption, acknowledgement of receipt, checksums, etc., could be built on top of `filetsf`.

We sometimes have the need to send sets of files, for processing as a unit. The simple approach to handling sets of files is to combine the separate files into a single file, using `tar`(1), or some other file archiving program. An alternative that might be useful would be to create a `getset` command, that would check to make sure that the complete set of files are available, retrieve them all, and run a command to process the files. This is something that is simple enough to do, and it may be a common enough operation to justify creating a command to do it.

There may be situations in which the `lpr/lpd` system or the LPD protocol is either not appropriate or not available. For example, there might be a need to transfer files over larger, public networks in a secure fashion, or through networks that do not allow LPD protocol traffic. It should be fairly easy to implement `filetsf` on top of a different method of transport, or to use multiple transports depending on the sender and recipient.

Conclusions

`Filetsf` seems to meet our needs for a basic file transfer system, and is an improvement over our past approaches to file transfer. While not providing extreme levels of security, it is a useful tool on our systems, and we expect to make fairly extensive use of it in the future. It was relatively easy to implement, and, if our needs change in the future, it should be a fairly straightforward task to replace `lpr/lpd` with some other transport mechanism providing greater reliability or security, while leaving the user interface unchanged.

Acknowledgements

The VM/CMS client implementation is due to the efforts and knowledge of Cameron McDonald.

Author Information

John Sellens is a Chartered Accountant and holds a master's degree in Computer Science from the University of Waterloo. He is currently Project Leader, Technical Services in the Data Processing department at the University of Waterloo where he is responsible for system administration and planning, and tool development for the administrative computing systems. John likes to provide as long a list of references as possible in each paper that he writes, whether or not he has actually read whatever it is he's referring to. John can be reached by mail at Data Processing, University of Waterloo, Waterloo, ON N2L 3G1, Canada, or electronically as jmsellens@uwaterloo.ca .

Availability

The current version of `filetsf` is available through anonymous FTP to math.uwaterloo.ca as 'pub/filetsf/filetsf.tar.Z' or through the author. Installation and configuration should be fairly straightforward at most sites. Like most freely available software, no support is available for `filetsf`, but bug fixes and suggested enhancements would be gratefully accepted.

References

[1] Anderson, Paul, "Managing Program Binaries In a Heterogeneous UNIX Network", *LISA V Proceedings*, San Diego, CA, October, 1991, pp. 1-9.

[2] Borenstein, N., and N. Freed, *MIME (Multipurpose Internet Mail Extensions): Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies*, RFC 1521, September, 1993.

[3] Campbell, Ralph, *4.3BSD Line Printer Spooler Manual*, Computer Systems Research Group, University of California, Berkeley, 1986.

[4] Colyer, Wallace, and Walter Wong, "Depot: A Tool for Managing Software Environments", *LISA VI Proceedings*, Long Beach, CA, October, 1992, pp. 153-159.

[5] Cooper, Michael, "Overhauling Rdist for the '90s", *LISA VI Proceedings*, Long Beach, CA,

October, 1992, pp. 175-188.

[6] Dagenais, Michel, et al, "LUDE: A Distributed Software Library", *LISA VII Proceedings*, Monterey, CA, November, 1993, pp. 25-32.

[7] Manheimer, K., B. A. Warsaw, S. N. Clark, and W. Rowe, "The Depot: A Framework for Sharing Software Installation Across Organizational and UNIX Platform Boundaries", *LISA IV Proceedings*, Colorado Springs, CO, October, 1990, pp. 37-46.

[8] McLaughlin III, L., ed., *Line Printer Daemon Protocol*, RFC 1179, August, 1990.

[9] Nachbar, Daniel, "When Network File Systems Aren't Enough: Automatic Software Distribution Revisited.", *Proceedings of the Summer USENIX Conference*, Atlanta, GA, June, 1986, pp. 159-171.

[10] Postel, Jonathan B., *Simple Mail Transfer Protocol*, RFC 821, August, 1982.

[11] Postel, J., and J. Reynolds, *File Transfer Protocol*, RFC 959, October, 1985.

[12] Sellens, John, "Software Maintenance in a Campus Environment: The Xhier Approach", *LISA V Proceedings*, San Diego, CA, October, 1991, pp. 21-28.

[13] Wong, Walter C., "Local Disk Depot – Customizing the Software Environment", *LISA VII Proceedings*, Monterey, CA, November, 1993, pp. 51-55.

[14] Zimmerman, Philip, *The Official PGP User's Guide*, ISBN 0-262-74017-6, MIT Press, 1995.

ACCEPTFILE(1) USER COMMANDS ACCEPTFILE(1)

NAME

acceptfile – retrieve or list files sent via sendfile

SYNOPSIS

acceptfile [*-a*] [*-c*] [*-d*] [*-l*] [*-m*] [*-n*] [*-o*] [*-p*] [*-r newfilename*] [*-s sender*] [*-t filetime*] [*-u recipient_user*] [*-v*] [*filename ...*]

DESCRIPTION

acceptfile is used to list files waiting to be picked up from the file transfer system (see *filetsf*(5)), or to retrieve (accept) files from the spool directory for use.

OPTIONS

-a all users – list files for all users (in conjunction with *-l*)

-c columns – list files in columns, for easy interpretation by other programs

-d delete – delete a particular instance of a file, requires *-t*

-l list – list the files that are waiting to be picked up

-m missing is ok – don't indicate an error if no matching file was in the spool

-n newest – accept the newest instance of a file

-o oldest – accept the oldest instance of a file

-p purge – actually remove files after they are accepted or deleted – without *-p*, files are renamed and removed later so that they can be retrieved in case of error (requires system administrator's assistance)

-r newfilename
rename – the file will be accepted into a file with *newfilename* rather than the name used by the sender

-s sender
select files sent by *sender* – comparisons are case insensitive

-t filetime
select files placed into the spool directory at time *filetime* – *filetime* is given as an integer number of seconds since the epoch, and is listed with *-l*

-u recipient_user
user – list files for user *recipient_user* (in conjunction with *-l*)

-v verbose – be a little noisier

filename
the name(s) of the file(s) to be listed or accepted

EXAMPLES

List the files waiting for you to accept:

```
acceptfile -l
```

Accept the oldest copy of a file named *blort*, sent by any user:

```
acceptfile -o blort
```

Accept the most recent version of a file called *blam* sent by *jsmith@mach1* and save it as the file *bloop*:

```
acceptfile -n -s jsmith@mach1 -r bloop blam
```

Accept a version of the file if one is waiting, and then process the file:

```
acceptfile -o -m blort
if ( ! -f blort ) exit 0
process_file blort
```

SEE ALSO

filetsf(5), sendfile(1)

BUGS

Perhaps there should be a -x "execute command" argument for a command to run afterwards, but that's harder, since we would have to really, really carefully renounce our extra permissions, so it's harder.

Perhaps there should be a *getset* command that will accept a list of files and let you know when everything is in place so that you can process all those files at one time.

AUTHOR

John Sellens, University of Waterloo

NAME

aixbe – printer backend program for use on AIX

SYNOPSIS

aixbe *-H options:recipient* [*-w width*] *spoolfile*

DESCRIPTION

aixbe is used in the */etc/qconfig* file on AIX machines to spool files that were sent to this machine. It processes its arguments, retrieves some additional information from the printer system, and then invokes *tsfif*(8) to do the actual work.

See *filetsf-config*(7) for information on use and queue configuration.

BUGS

The AIX printing system.

SEE ALSO

filetsf(5), filetsf-config(7)

AUTHOR

John Sellens, University of Waterloo

NAME

filetsf-config – how to configure the file transfer system

DESCRIPTION

Some things are obvious, some aren't.

USERS AND GROUPS

If you want to receive files on a machine, you have to add a group called *filetsf* to the machine, and you need to add the user *daemon* (or *lpd* if you're on AIX) to the *filetsf* group.

SPOOL DIRECTORY

You'll need to create the spool directory (check your configuration options for the name, at Waterloo we use */software/filetsf-1/spool/.* as the spool directory). It should be owned by *daemon* (or *lpd* on AIX), in group *filetsf*, and be mode 750 with the setgid set.

PRINTCAP ENTRIES

These instructions are for machines with normal printing systems. If you have an AIX machine, see the next section.

You need to set up a print queue called *tsf-machine* for every *machine* you want to send files to. Your */etc/printcap* entry should look something like this:

```
tsf-machine:\
    :mx#0:\
    :lf=/software/lpr/spool/tsf-machine/log:\
    :rp=filetsf:\
    :rm=machine:\
    :fx=f:\
    :sd=/software/lpr/spool/tsf-machine:
```

On each machine you want to receive files on, you need to set up a print queue called *filetsf* with a */etc/printcap* entry something like this:

```
filetsf:\
    :lp=/dev/null:\
    :af=/dev/null:\
    :sd=/software/lpr/spool/filetsf:\
    :lf=/software/lpr/spool/filetsf/log:\
    :if=/software/filetsf-1/servers/tsfif:\
    :fx=f:
```

You'll need to make sure your sending machines have permission to send to the receiving machines, either through the */etc/hosts.equiv* file or through appropriate entries in your */etc/printcap* file.

AIX NOTES

I hate, hate, hate the AIX printing system. Everything about it.

Groups and Users

The printing software runs as user *lpd* so the spool directory needs to be owned by *lpd* and not *daemon* as on other systems. You will need to add *lpd* to the *filetsf* group, because otherwise *tsfif*(8) will not be able to set the setgid bit on the directories it creates, and the group of the directories will not propagate.

aixbe The program *aixbe*(8) is an AIX printing backend, and retrieves the necessary information from the printing system database, and calls *tsfif*(8) appropriately to do the actual work.

Loop Around

It seems that AIX expects local printer devices to be mode 666, and so if you configure the file transfer queue as you would expect, *tsfif*(8) ends up running as the local user (if the user is sending a file to someone else on the same machine), which just doesn't work. Pretty silly if

you ask me, but no one did.

The way to deal with that is to set up a local **tsf-machine** queue that prints to a remote **filetsf** queue on the same machine. But, AIX is too smart (ha, ha), and notices if the remote printer host is the same as the hostname of the local machine, and does something stupid if so (i.e. it tries to short circuit the process or something). So, use a different form of the machine name to fool the brilliant AIX printing software e.g. if the hostname is a fully qualified name, use an unqualified name in the *qconfig* file, and vice-versa. Note that the final queue has to be configured as a local queue, because AIX gives different arguments to the backend programs depending on whether or not the queue is local or remote. Strange but true.

Sample Configuration

This seems to work for us:

```
filetsf:
        up = TRUE
        device = dfiletsf
dfiletsf:
        backend = /software/filetsf-1/servers/aixbe
tsf-mach1:
        s_statfilter = /usr/lpd/aixshort
        l_statfilter = /usr/lpd/aixlong
        up = TRUE
        device = rtsf-mach1
        host = mach1.uwaterloo.ca
        rq = filetsf
rtsf-mach1:
        backend = /usr/lpd/rembak
```

AUTHOR

John Sellens, University of Waterloo

NAME

filetsf – file transfer system internal details

DESCRIPTION

The **filetsf** system is built using *lpr*(1) and *lpd*(8) as the transport mechanism. This fact is cleverly hidden from the user via the *sendfile*(1) and *acceptfile*(1) commands, and the *tsfif*(8) input filter.

LPR/LPD ENCODING

The following information is encoded in the lpr/lpd protocol as follows:

Sender in the lpr user/host

Recipient Machine

Each machine that can receive needs a separate queue, named for the machine, with a prefix of tsf-, as in tsf-mach1. The receiving queue is normally named filetsf, which makes it possible to send files to someone on the same machine.

Recipient User

Encoded in the **-C** (class) option to *lpr* and is limited to 17 characters, so that it and the options (if any) will fit within the 31 character limit imposed by the lpd protocol.

Filename

The real or intended name of the file is encoded using the **-J** (job) option to *lpr* and is limited to the 99 character limit imposed by the lpd protocol.

File Transfer Options

The file transfer system encodes its options in the **-C** (class) option to *lpr* and is limited to 10 characters, so that it and the recipient will fit within the 31 character limit imposed by the lpd protocol.

OPTIONS

Not all options have been defined yet, but additional ones should be considered for things like expiry time, and so on. The options are encoded by summing their numbers (note that they're powers of two).

```
OPT_MAILSENDER      (1)   /* mail sender on delivery */
OPT_MAILRECIPIENT   (2)   /* mail recipient on delivery */
OPT_REPLACE         (4)   /* ignore all but newest */
```

QUEUE FILENAME ENCODING

Once the file is received on the destination system, *tsfif* stashes it in a file, encoded like this:

/spooldir/recipient/name/sender@host/time

where time is the time in seconds since the epoch.

CONFIGURATION

See *filetsf-config*(7).

SEE ALSO

RFC 1179 Line Printer Daemon Protocol

AUTHOR

John Sellens, University of Waterloo

NAME

ftcheck – do a sanity check on the filetsf system

SYNOPSIS

ftcheck

DESCRIPTION

ftcheck tries to sanity check the **filetsf** system, by checking the *filetsf* group, the permissions and ownership of the spool directory, etc.

ftcheck should be run periodically from *cron*(8)

BUGS

Or should I say potential enhancements?

Permissions
: It doesn't actually check the permissions.

Log Rolling
: There should be some mechanism that rolls the *aixbe*(8) log file on AIX (we use our *roller*(1) command at Waterloo). Presumably everyone is already rolling their *lpd*(8) logs.

Queue Checking
: There should be some mechanism that notices if the **filetsf** print queue is blocked (our print system notices this at Waterloo).

NIS
: It doesn't handle NIS'd group files.

SEE ALSO

filetsf(5), ftcleanup(8)

AUTHOR

John Sellens, University of Waterloo

NAME

ftcleanup – cleanup old files in the filetsf system

SYNOPSIS

ftcleanup

DESCRIPTION

ftcleanup wanders the *filetsf* spool directory looking for old and deleted files.

ftcleanup should be run periodically by *cron*(8) (usually daily).

BUGS

It might be better if the *filetsf* system was locked while *ftcleanup* was running, but the only way I can think of to do that is by running it through the *filetsf* queue itself, but that would be kind of weird.

It would be nice if we sent mail to the sender and recipient of unclaimed files, but I'm too lazy to do that for now.

It would be nice if *ftcleanup* removed old versions of files flagged as being replaced by newer versions, whether or not the older files had been retrieved yet.

SEE ALSO

filetsf(5), ftcheck(8)

AUTHOR

John Sellens, University of Waterloo

NAME

sendfile – queue a file for sending to a remote machine/user

SYNOPSIS

sendfile *-h hostname* [*-j jobname*] [*-m*] [*-M*] [*-r*] [*-u user*] [*-v*] [*file ...*]

DESCRIPTION

sendfile is used to submit one or more files to the file transfer system for transfer to a remote machine.

If no files are given on the command line, nothing is sent; a filename of - means to send the standard input.

OPTIONS

-h hostname
: Send the file(s) to *hostname.* Required.

-j jobname
: Label the file with the name *jobname*. Defaults to the file's name. Required if standard input is being sent.

-m
: Send mail to recipient on a file's arrival

-M
: Send mail to sender on a file's arrival

-r
: Replace any older copies of this file (in the spool directory) with this one.

-u user
: Send the file(s) to *user* on *hostname*. Defaults to the sender's name.

-v
: verbose – be a little noisier

SEE ALSO

acceptfile(1), filetsf(5)

BUGS

Doesn't know how (yet) to validate hostname, so if the host isn't configured for file transfer, you get an obscure message from lpr complaining about a strange print queue.

It would be nice if multiple *-h* options were accepted, and the files were sent to each host in turn.

AUTHOR

John Sellens, University of Waterloo

NAME

tsfif – lpd filter for transferred file reception

SYNOPSIS

tsfif *-C opts:recipient -h srchost -J name -n sender* [*-v*] <*other lpr filter args*>

DESCRIPTION

tsfif is an lpr print filter, used to store files sent by *sendfile*(1) in a spool directory for later retrieval via the *acceptfile*(1) command.

Put this in */etc/printcap* on the transfer reception queue, and this will take the received file and put it in the file transfer spool area. See *filetsf-config*(7) for an example of use.

OPTIONS

The options are the normal lpr print filter arguments; see the *filetsf* documentation for more details.

SEE ALSO

acceptfile(1), sendfile(1), filetsf(5)

AUTHOR

John Sellens, University of Waterloo

Patch Control Mechanism for Large Scale Software

Atsushi Futakata – Central Research Institute of Electric Power Industry (CRIEPI)

ABSTRACT

Applying patches to large scale software is often difficult because unofficial patches and user modifications conflict with any "official" patches. Version control systems such as RCS[1], CVS[2], and configuration management[3,4,5] are useful solutions for this problem when the baseline of the software is fixed. However, an official patch that is developed externally changes the baseline and any local changes based on this become obsolete. Thus we must re-apply various unofficial patches and modifications, identify the causes of conflict, change or remove patches, and repeat the patch and unpatch operations.

This paper presents a mechanism for (1) managing versions of a software package based on patches, (2) automating the application of unofficial patches and modifications by the user, and (3) rebuilding the package using file versions instead of timestamps. Using this mechanism, it becomes easy to apply patches and re-build software.

Introduction

We have spent a lot of time installing and patching large scale software packages such as the X11 Window System, TeX, etc. Installation of new software involves checking storage space, reading documentation and setting various configuration files correctly. This can be a non-trivial task even if the platform is officially supported. If the platform is not supported, installation becomes more complicated because changes to the source code may be required and tools such as *Configure* are not applicable. Thus software porting systems represented by the FreeBSD ports system[6] appear and become to support the installation task.

Applying patches poses another difficult problem: If only official patches are applied to an officially supported platform, the task is usually easy because the patches are well managed and cause no conflict. However an unsupported platform requires source code changes which often conflict with an official patch. Furthermore, the user may require many useful, unofficial patches. These may be patches for emergency security, localization (e.g., japanization), machine/OS-dependencies or various extensions, such as Tcl/Tk has. Those patches may also conflict with official ones. The reason for the conflict is a lack of version management facilities for distributed development. This conflict usually necessitates the following operation:

- Remove all unofficial patches and apply the official one,
- Re-apply the unofficial patches and user modifications. If reject files are generated, the unofficial patch must be fixed or removed,
- Rebuild the software. This can take a long time because the above operations may cause unnecessary changes to timestamps.

Configuration management systems such as Aegis[7], CMS/MMS[8] are useful for version control and building software for multi-user development. They target the continuous development of the software and manage products based on a current baseline, that is a reference version of software on which each member of developing team fixes bugs and develops new functions. After each task is complete, all modifications are integrated and the modified source code becomes a new baseline for succeeding development. This baseline approach is useful for inhouse development teams.

However, an official patch is delivered outside of a user's control and it only changes the baseline. All modifications based on the previous version of the software then become obsolete. Thus if the user wants to apply a new official patch, all other patches and modifications must be rearranged and re-applied after the official one is applied.

In future, self-adaptive software agents or automatic programming from very high level specifications may solve the problem but, for the present, we have no silver bullet. Thus, in order to solve the above problem and support patch application, this paper proposes a patch control mechanism which has the following features:

- version management of the whole package, including individual files and patches,
- management and control of patch application order,
- assistance with patch/unpatch operations and patch modification,
- software rebuilding according to an individual file version rather than a timestamp.

Classification of patches

Unofficial patches may be generated by different people based on different baselines. Unified management of these patches can be difficult and confusing. In this paper, we classify patches into the following three types and treat each differently.

official patch
A patch that is authorized by and distributed from the software developer/maintainer. The latest official patch number is the official software version and we call it the patchlevel.

unofficial patch
A patch/extension that is widely distributed but is not an official patch. An unofficial patch may be applied in various directories with various *patch*(1) options.

modification
A change made by the local user, which includes editing files, fixing bugs, changing configuration files, etc.

System Overview

This section presents an overview of the system which is an implementation of the patch control mechanism. This includes; (1) management of the three types of patch, (2) control of patch application, and (3) rebuilding of the software. The components *VM (Version Manager)*, *PM (Patch Manager)*, and *BM (Build Manager)* implement the three functions respectively. Figure 1 shows the components and the relation among them.

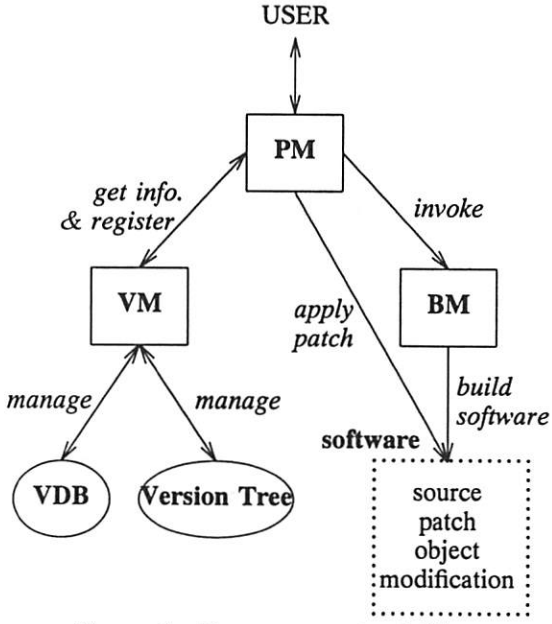


Figure 1: The components of this system

The VM manages information in the *VDB (Version Database)* and the *version tree*, which records information about version control for updating and rebuilding of the software. The VDB records the location of the patches and the versions of individual files. The version tree records the application order for unofficial patches and modifications at each patchlevel. When a new patch arrives, the user adds the patch to the VDB using the VM. If the user wants to apply this, and the result is successful, the VM registers the sequence of patches actually applied, to the version tree.

The PM (Patch Manager) controls the patch/unpatch operations and the building of the software according to the version tree. In this system, all operations, including editing a patch file, applying a patch, and building a package, are performed via the PM, and the result of the operation is reflected in the VDB and the version tree.

When a user applies a new official patch, the PM tries to apply unofficial patches which were applied to the last version of software. If one of the patches is rejected, the PM notifies the user. The user may then remove or edit this patch and continue the job. After the job finished, the PM returns the result of the patch application and the VM revises the VDB and the version tree.

The BM (Building Manager) is an extended *make* command, invoked from the PM to build a target according to the version of file instead of its timestamp. Because a new official patch forces patch/unpatch operations of unofficial patches and modifications, the timestamp of a file may change even if the contents is not altered. The VM registers the version of the newly generated target with the VDB and the version tree.

This system manages several packages at once by referring to the **pcm** file. An entry of the **pcm** file has the following form:

application_name:top_directory:patch_option

Application_name is an identifier to be used for selecting a package. *Top_directory* is the directory where official patches are applied. *Patch_option* species an option to the *patch*(1) command. For example, the entry for X11R5 (X11 Window System, Version 11, Release 5) becomes[1]:

```
X11R5:/X11R5:-p -s
```

This means that the following command is needed to apply the patch.

```
% cd /X11R5
% patch -p -s <foo.patch
```

Version Management

In this system, changes of source codes and the source code itself are managed separately to make

[1]Because of the limitation of line width, we denote the location of the X11R5 package as `/X11R5` in the following examples. The actually location is `/staff/src/X11R5` in our site.

patches independent of the baseline specified by an official patch. Thus each unofficial patch and modification has a separate version to avoid conflicts which may occur with any new official patch. The versions of these patches are managed using RCS.

The VM manages the patch information, including the location of patches, versions of patches themselves and the history of patch application. The VM performs the following functions:

- generates the specified version of a software package or a file by applying patches automatically,
- registers/deletes/updates a patch,
- creates/updates a modification from the differences between a modified file and its original,
- maintains versions of files, which are determined by the patches which actually cause change.

Information managed by the VM is recorded in the VDB and the version tree. The VDB consists of the locations and the versions of patches and the versions of the individual files. The version tree describes the order of patch application required to make the specified version of software, and which version of each patch should be applied.

The version of the software package itself is represented by a path of the version tree. For example, `#3:@1.2,@2.1,@3.2:$1.2` means that the version is generated by application of an official patch #3, unofficial patches `@1.2`, `@2.1`, and `@3.2`, and a modification `$1.2` in order. The following section describes the contents of the VDB and the version tree.

```
#26
#1:/X11R5/fixes/fix-01
#2:/X11R5/fixes/fix-02
#3:/X11R5/fixes/fix-03
#4:/X11R5/fixes/fix-04
```

Figure 2: A part of .official file for X11R5

Version Database

The VDB consists of the four files.

.official

`.official` contains the current patchlevel and the locations of official patches. The first line is the current patchlevel of the software. The following lines contain a patch identifier (which is used in the version tree), and a corresponding patch location. Figure 2 is a sample of a part of a `.official` file. `#26` in line 1 means that the current patchlevel is 26. The lines 2-5 specify the location of each official patch. For example, line 2 means that the location of the official patch `#1` is `/X11R5/fixes/fix-01`.

.unofficial

`.unofficial` contains the locations of unofficial patches and the information required to apply them. Each entry of this file has the following form;

id:location:place:option

Id is the unofficial patch identifier which is used in the version tree. *Location* is the location of the unofficial patch. *Place* is the directory in which the patch is applied, and *option* is the *patch*(1) options. In general, there is no standard method for applying unofficial patches, and this is a reason for the *place* and *option* fields. Figure 3 shows a part of `.unofficial` for X11R5.

```
@1:/X11R5/fixes/Xaw-p1:/X11R5:-p0
@2:/X11R5/fixes/Xsi-p1:/X11R5:-p0
@3:/X11R5/fixes/Xwchar-p1:/X11R5:-p0
@4:/X11R5/fixes/Xaw-p2:/X11R5:-p0
```

Figure 3: A sample of .unofficial file for X11R5

Versions of the patches are managed by RCS and the RCS file for each patch is located in *directory of location/RCS*. A user can edit the patched files themselves instead of the patch because it is almost impossible to edit the patch directly. Changes to the files are reflected in the patch by the following process:

- choose the version of the software and the target patch to be edited. For example, we assume that the patch is `@3.1` which changes two files, `foo.c` and `bar.c`, and the version is `#3:@1.2,@2.1,@3.1`,
- apply the sequence of patches which should be applied in this version before applying the target patch. After that, make a copy of the patched file and apply the target patch. In this example, first, the VM applies `@1.2` and `@2.1` to the software whose patchlevel is #3. Next, the VM makes copies of `foo.c` and `bar.c` with an extension *.prev*. Then, the VM applies `@3.1` to the software,
- after editing the patched files, make a new patch by running *diff*(1) against the files of which the VM made copies in the last step. In this example, the two diff files between `foo.c/bar.c` and `foo.c.prev/bar.c.prev` are concatenated to a new patch, whose path name is the same as `@3.1`.
- check in the new patch using *ci*(1) . In this example, the VM runs the following command:

```
% ci -r2.1 location of @3
```

in which 2 of `2.1` is the new version number for the patch `@3`.

The VM normally uses only the release number of RCS. Thus a patch with version *N* has a revision *N.1* in the RCS file. For example, when applying the version 3 of the patch `@1` in the Figure 5, the following commands are needed:

```
% co -l -r3.1 /X11R5/fixes/Xaw-p1
% cd /X11R5
% patch -p0 </X11R5/fixes/Xaw-p1
```

.modification

`.modification` contains the locations of user modifications. There is at most one modification file per directory and the result of editing the source is reflected in the modification file in the same way as unofficial patch files. Figure 4 shows a part of `.modification`. The first field is the identifier of modification and the second specifies the location of the modification.

```
$1:/X11R5/mit/config/config.patch
$2:/X11R5/mit/lib/Xt/Xt.patch
```

Figure 4: A sample of .modification file for X11R5

.f_ver

`.f_ver` contains the version history of the source and object files. In this system, a file has two different forms of version. One is the strict version which is indicated by the software version. The other is the historical version which indicates the history of changes by patches. The historical version is used instead of the file timestamp when the software is rebuilt. For example, the strict version of file `foo` is indicated as:

```
foo.#3.{@1.1,@2.1,@3.2}.$1.2
```

where `#3` means the official patchlevel is 3 and *@N.M* means that the applied unofficial patch identifier is *N* and the version of the patch itself is *M*. `$1.2` means that the version of a modification to `foo`. The historical version has the following form:

```
foo:#1,#3:@1.1,@2.1:$1.2
```

This means that `foo` is changed by the official patches `#1` and `#3`, unofficial patches `@1.1` and `@2.1`, and user modifications with version `$1.2`.

`.f_ver` must exist in all subdirectories of the software source tree. An source/object entry in this file is updated as follows:

- if a patch is applied to the file, the identifier of the patch is added to the entry,
- if the version of a source file differs from the object file, after making the object, the object is given the same version as the source. If multiple sources exist, e.g., linking *.o files, the versions of the sources are merged and becomes the version of the object because it is made under the effect of patch applications to the sources. This method is described in the section **Make Command**,
- editing the file changes the version of the modification in the entry.

If the file in the VDB is a symbolic link, the VM follows the link and updates the location of the file to be the real location.

Version tree

The version tree manages the software version and describes the application order of unofficial patches and modifications at each patchlevel. The version tree includes applied unofficial patches and modifications only. Figure 5 shows the concept of the version tree. *#N* is the patchlevel and *@N.M* and *$N.M* are the unofficial patch identifier and the modification identifier to be applied. A conflict between unofficial patches causes branching or modification of a patch.

The `.vtree`, which is located in the top directory of the software, records the version tree as the collection of the following form;

official_id:unofficial_ids:modifications

For example, the path **A** in figure 5 is described as "**#1:@1.1,@2.1,@4.1:$1.1,...**" and the path branched from `@1.1` is described as "**#1:@1.1,@3.1, @4.2,...**".

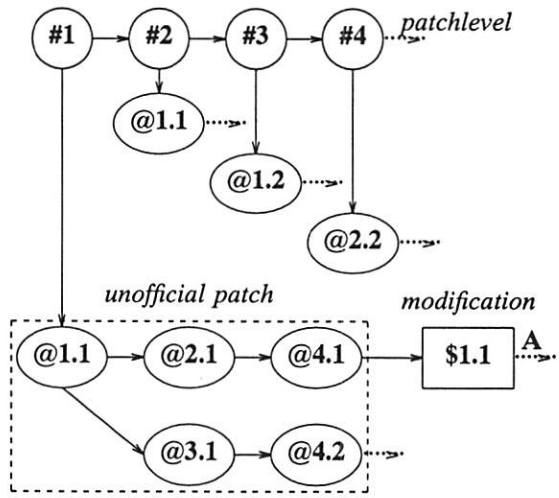


Figure 5: The concept of version tree

Patch Control Mechanism

This section describes the patch control mechanism based on the version tree. The PM stores source files to which only official patches are applied. The unofficial patches/modifications are applied on demand when editing the latest version sources, rebuilding the software, etc.

The PM provides an asynchronous way to apply a patch or to rebuild a software package by exchanging information with the user via e-mail. Once the PM is invoked, the PM reports conflicts or compilation failure to the user via e-mail. After the user edits files or abandons the patch, the user only sends a simple command with the file contents if it is needed. The PM accept the following commands:

- **edit** [*file* | *id*] *(ver)*
  edit the *file* or a patch whose identifier is *id* in the software version *ver*. In the PM interface,

a user can specify the *ver* in the form given in the version tree, or as the entry number in the version tree which the user can obtain by sending the command **scan -v**. This causes a change to an unofficial patch or a modification.

- **add [-o |-u]** *patch_info*
  register an official/unofficial patch with the *patch_info*. The *patch_info* has the same form as in `.official` or `.unofficial` without the identifier. The identifier is added by the VM.
- **scan [-v |-o |-u |-m]**
  show the contents of `.vtree`, `.official`, `.unofficial`, or `.modification`.
- **show** *id (ver)*
  show the contents of patch whose identifier is *id* in the software version *ver*. If the *ver* is omitted, **show** shows the patch that has been applied in the current version of software, or the latest version of the patch if the current version of software does not include this.
- **apply** *id (ver)*
  apply a patch whose identifier is *id*. If the patch is an official patch, a user does not need to specify the *ver*. If the patch is an unofficial patch or a modification, the user must specifies the *ver* as the full path form. Processing of the **apply** function is described later.
- **make [-T]** *(ver)*
  rebuild a software package whose version is *ver*, or the current version if the *ver* is omitted. If a user wants to rebuild the package according to timestamps, **-T** option is necessary.
- **grep [-o |-u |-m |-a]** *string*
  grep *string* from official patches, unofficial patches, modifications and all patches.

When **apply** or **make** is issued, the PM interacts with the user. When applying a new official patch, the PM performs the following actions:

1. apply the official patch.
2. make a sequence of unofficial patches to be applied. The PM extracts the last sequence from the version tree and notifies the user via e-mail. The user can remove several patches or change the application order. The user then replies by e-mail.
3. apply unofficial patches and any modifications in order. If no reject files are generated by *patch*(1), step (3) is done. Otherwise the PM suspends the job and sends the patched file with the rejected part embedded at a suitable place to the user via e-mail. The user must decide either to edit and apply this patch or to abandon this. If the user decides to abandon, the PM strips this patch from other files to which the patch was applied and the PM applies a next patch. The user can edit it before replying to the PM. In this operation, the user can refer to the whole patch by sending a command to the PM. When the all replies are gathered, the PM replaces the parts of the patch and reflects the change in the RCS file using *ci*(1). After that, the VM tries to apply the next patch.
4. create a new entry in the version tree. If user wants to rebuild the software, the PM invokes the BM.
5. unpatch all the unofficial patches and the modifications.

When applying a new unofficial patch, the user should specify the version tree entry which includes the new patch. This new version tree entry is registered if the application process has terminated successfully. The other operations are similar to the above case for official patches.

If reject files are generated, the PM sends e-mail to the user per reject file. Figure 6 is a sample of the e-mail, which notifies that application of unofficial patch `$1.1` has been failed in `sun.cf`[2]. The body of e-mail is the contents of `sun.cf` and the rejected file is embedded in it.

```
To: futakata
Subject: REJECTION $1.1: sun.cf
From: PCM <pcm@denken.or.jp>

<Contents of sun.cf continue.>
      ......
```

Figure 6: A head of e-mail which notifies the rejection

After receiving the e-mail, the user can choose from the following three actions:

- **replace**: replaces a part of patch/modification by this file,
- **delete**: deletes this patch from the version tree entry,
- **abort**: aborts the job, recovers the software and terminates the PM.

If the user chooses **replace**, the user must reply the e-mail like Figure 7 to the PM after editing the file in the original e-mail. The first line of the mail body is the action that the user chosen, and the rest of the body is the file to be replaced. If the user chooses **delete** or **abort**, only the action is needed in the body. After receiving all replies from the user, the PM extracts a new patch from the replies by the same manner described in the explanation of `.unofficial`, and applies the new patch instead of the original patch and revises the version of the patch.

[2]We abridge the file name in the subject field of Figure 6 for convenience of display. The subject of real e-mail contains the full path name of the file.

```
To: pcm@denken.or.jp
Subject: RE: REJECTION $1.1: sun.cf
From: futakata@denken.or.jp

replace
<Contents of new sun.cf continue.>
    ......
```

Figure 7: A head of reply mail with the action **replace**

Make Command

The PM often causes needless changes to timestamps and confuses timestamp-based traditional *make*(1) commands. Thus we developed the BM, which is based on the *GNU make* command, to perform actions according to the historical file version, i.e., when the historical versions of source files are not included in the version of the object, the BM rebuilds the object. After that, the BM revises the version of object by the merged version of sources because the object is made under the effect of all patches applied to the sources.

To compare versions, the BM treats the version as an ordered sequence of patches. Thus a version *A* includes a version *B* if and only if all patches in the *B* is appeared in the *A*, and an unified version of *A* and *B* does not causes inconsistency of the application order. For example, the version `#1:$1.1,$2.1,$3.1` does not include the `#1:$3.1,$1.1`.

After building objects, the BM sends the result to the VM and the VM updates the historical versions of the objects.

Experiment

We have tested this system for applying patches to the X11R5. In this experiment, the number of official patches is 26 and the number of unofficial ones is 12 including internationalization patches for libraries Xaw, Xsi and Xwchar. We also modify configuration files in the `/X11R5/mit/config` directory and several files in `/X11R5/mit/lib/Xt`. Figure 8 is a head part of the version tree of this experiment.

After the official patch `#3` is applied, 3 unofficial patches are issued and we apply them and rebuild the X11R5. There is no problem in this phase. When the `#6` is applied, the internationalization mechanism of X11R5 is changed and `@1.1` and `@2.1` become obsolete. The PM then sends e-mail telling us that reject files are generated. We remove the patches and do not rebuild X11R5. After `#8`, a new unofficial patch `@4.1` is issued and we rebuild after applying `@4.1`.

This experiment shows that it is easy to control the patch/unpatch operations and to find conflicts between official patches and unofficial ones.

```
#0::$1.1,$2.1,$3.1,$4.1
#1::$1.1,$2.1,$3.1.$4.1
#2::$1.1,$2.1,$3.2.$4.1
#3:@1.1.@2.1,@3.1:$1.2,$2.1,$3.3,$4.1
#4:@1.1,@2.1,@3.1:$1.2,$2.1,$3.3,$4.1
#5:@1.1,@2.1,@3.1:$1.2,$2.1,$3.3,$4.1
#6:@3.1:$1.2,$2.1,$3.3,$4.1
#7:@3.1:$1.2,$2.1,$3.3,$4.1
#8:@3.1,@4.1:$1.3,$2.1,$3.3,$4.1
```

Figure 8: A part of the version tree for X11R5

Conclusion

In this paper, we presented a control mechanism for applying patches and confirmed that this system works well. However, even if the patches are successfully applied, rebuilding the software still takes a long time. In order to reduce the rebuilding time, we have tried to develop a mechanism for analyzing the dependency between makefiles and a distributed make mechanism based on it.

Availability

The system which we presents in this paper will be available via WWW from http://www.denken.or.jp/people/cirl/futakata.

Acknowledgements

I would like to thank Paul Anderson at the Univ. of Edinburgh for his useful advice on drafts of this paper. My thanks also go to Yasusi Sinohara and Shouichi Matsui at the CRIEPI for their suggestion in development of this system.

Author Information

Atsushi Futakata graduated from the Tokyo Institute of Technology at Tokyo in 1987. He has made studies of automatic programming and management of distributed information system in the Central Research Institute of Electric Power Industry. He is now working on the CSCW environment for researchers.

Reach him via mail at: Information Science Department, Communication and Information Research Laboratory, CRIEPI, 11-1 Iwado-kita 2-chome Komae-shi Tokyo 201, JAPAN

His e-mail address is futakata@denken.or.jp.

References

[1] Tichy, W. F.: "RCS - A System for Version Control", *Software-Practice & Experience*, Vol. 15, No. 7, 1985.

[2] Berliner, B.: "CVS II: Parallelizing Software Development", included in the *CVS ver.1.3* package, 1995.

[3] Bersoff, E.: "Elements of Software Configuration Management", *IEEE Transaction*

of Software Engineering, Vol. SE-10, No. 1, 1984.

[4] Whitgift, D.: *Methods and Tools for Software Configuration Management*, John Wiley and Sons, 1991.

[5] Eaton, D.: "comp.software.config-mgmt FAQ version 1.12", posted to the `comp.software.config-mgmt`, 1995.

[6] Palmer, G. and Hubbard, J. K.: "The FreeBSD Ports FAQ file version 1.1", included in the *FreeBSD rel.2.0.5* package, 1995.

[7] Miller, P.: "Aegis: A project Change Supervisor User Guide", included in the *aegis ver.2.1* package, 1993.

[8] DEC: *CASE Environment of DEC: COHESION - COHESION handbook rev.1*, DEC, 1991.

From Twisting Country Lanes to MultiLane Ethernet SuperHighways

Stuart McRobert – Department of Computing, Imperial College, London

ABSTRACT

This paper describes a slightly different approach to solving network capacity problems between workstations and servers by significantly increasing the number of conventional Ethernet interfaces on each server from just a few to typically a dozen or more. So rather than installing a single faster network backbone (e.g., FDDI, ATM, Fast Ethernet, etc.) to carry all the traffic to and from the servers, coupled with some form of step down hubs to connect to the local workstation Ethernets, our approach bypasses the backbone completely and brings many local Ethernets directly to each of the servers (typically Sun Sparc Station 10s or 20s). This technique has worked very well for our size of operation with several file and CPU servers, 50+ workstations and around 100 X-terminals, with still room for some further expansion too.

Over the past year this approach has been very successful in our main teaching laboratories, significantly reducing network congestion and providing many more well connected networks to support both existing and additional workstations and X-terminals, yet with fewer clients per network, so easing local network contention problems. This, coupled with enhancements to the workstations and servers themselves, has yielded significant performance improvements all round and made for much happier and contented users.

Early Days – Twisting Country Lanes

A long time ago a colleague and I very carefully installed a pair of VAX 750s as the first hosts on our new Ethernet – a thick yellow heavy coaxial cable that ever so gently snaked its way around under the computer room floor – such care with a networking cable was probably never shown again! But users soon discovered how easy and convenient a rich set of new remote access commands were to use, e.g., *rcp*, *rlogin*, and *rsh*, and just how amazingly fast they could now transfer data between hosts. Meanwhile local file transfers successfully moved from the *uucp* tty port based era (cf. countryside foot paths) to this new amazingly quick single Ethernet (cf. a quiet single country lane). Incidentally *uucp* soon fought back for a while by offering queued user file transfers over Ethernet, which still appealed to some users.

Demand for Ethernet connectivity from research groups quickly became virtually unstoppable, almost like the modern day rush to get onto the *Internet* – everybody wanted to be connected. Fortunately for us there was just one moderating factor – cost.

Meanwhile the capacity of the network was at that stage never considered to be an issue, after all Ethernet had 10 Mbps bandwidth compared with only a few 19.2 Kbps tty circuits used before – capacity was almost considered to be infinite. However, as the single thick Ethernet cable began to spread, snaking its way out of the computer room and up the building, concerns were soon raised about its vulnerability to both physical and electrical damage. These were soon laid to rest with the installation of network repeaters on each floor, but fortunately this never became a real problem (Figure 1). The network was also extended via a bridge across campus, complete with our original and officially registered Class B IP network number, although fun and games

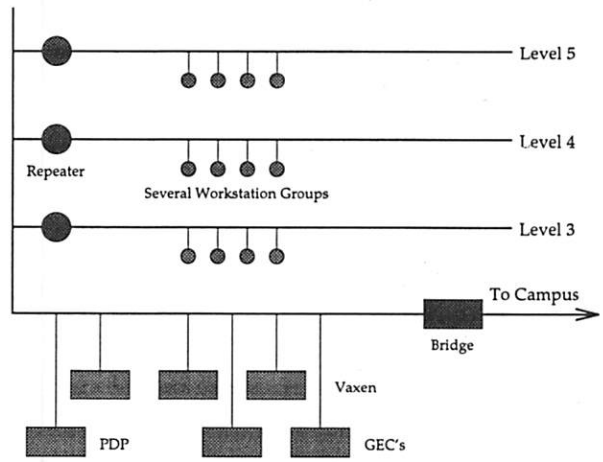


Figure 1: Early Departmental Ethernet (circa mid-80s)

were played with the netmask, eventually settling on a rather interesting 7/9 bit split which although politically acceptable caused no end of grief with various pieces of software. The bridge was a wise investment in terms of providing a surprising degree of protection and isolation from some rather strange and otherwise campus wide networking disasters. It

was however rather slow at forwarding packets (an issue we will return to later), but for now this wasn't much of a problem or concern since most host interfaces were also rather slow too.

With the Department's research groups successfully networked, our attention now turned to advancing teaching facilities and central services. After great debate and a lengthy search, a new powerful twin CPU Gould Powernode 9082 was purchased to act as our new central server. This system was great at handling I/O and since it was also very much more powerful than any of our other computers (both then or for the next few years) it soon took over nearly all central services. However, it did become a classic single point of failure, something that strongly influenced our later drive towards a far more distributed and fault tolerant or at least fault limiting approach. Ten 4 MB Sun 3/50 disk less student workstations and a small Sun 3/160 file server chiefly for Yellow Pages (YP, now NIS) and Network Disk support (ND was needed for booting workstations, root and swap areas in those days) were purchased for the undergraduate teaching laboratory.

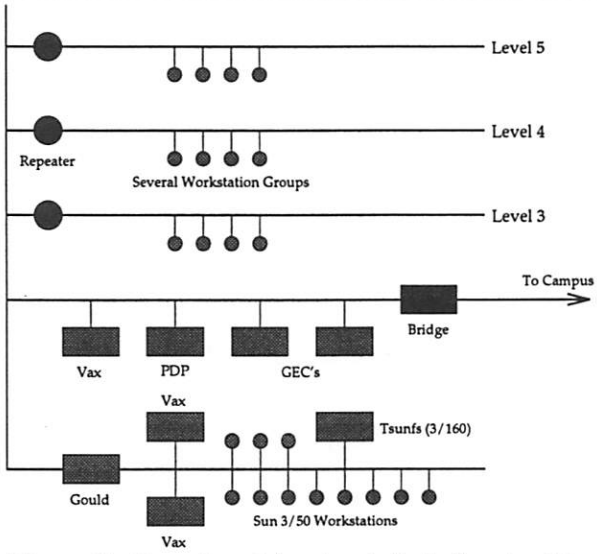


Figure 2: Departmental network including teaching (circa 1987)

However, for the first time concerns were raised about Ethernet's capacity to handle large traffic levels typically generated by many disk less workstations, especially due to all the additional paging and swap traffic produced because of a lack of enough workstation memory. So it was wisely decided to introduce a new teaching network rather than risk overloading the existing network any further. Probably the most interesting and significant choice at that time was the decision to connect this new network via a second Ethernet interface on the new central server, so creating the first of many multi-homed hosts (Figure 2). The Gould was superb at handling I/O and could easily and efficiently handle the extra traffic and still make good use of having access to twice the network bandwidth, in fact it later gained a third Ethernet interface and still coped well.

Over the next few years the number of disk less teaching workstations more than doubled with many additional Sun and HP workstations, along with several multi-purpose severs often with twin Ethernet interfaces, for both CPU and file serving work (Figure 3).

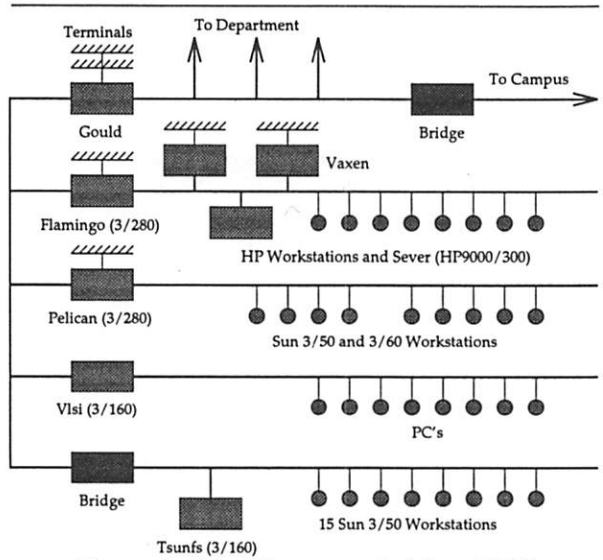


Figure 3: Teaching network (circa 1989)

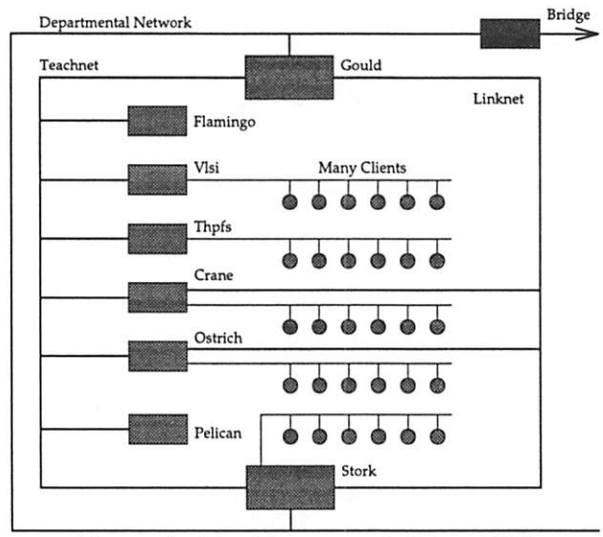


Figure 4: Teaching network (circa 1990)

Such growth continued especially as the use of glass ttys dwindled and graphical workstations proved highly successful, but traffic levels on the teaching networks rose at an alarming rate, coupled with high network collision levels during the ever lengthening peak periods. A full discussion of the problems faced at that time is outside the scope of this paper,

but can be found in [SunUG'91]. However it is worth noting the key changes in network topography carried out at this time to better cope with the ever rising traffic levels (Figure 4).

A new server (see *Stork* in Figure 4) with four network interfaces was introduced and along with *Crane* and *Ostrich* were the first Sparc servers purely dedicated to serving, i.e., they supported no user logins, and were locally known as Network Support Nodes or NSNs for short. They provided the workstation users with a much better response since their CPUs were never tied up with user jobs and had fairly good network connectivity to the other servers. For now they also had a speed advantage over the earlier generation of workstations, something that wouldn't last for long. Note that the bridge used earlier to help ease network congestion has been removed, since it was actually found to cause more of a network bottleneck than a help, since it was unable to forward packets at anything like network speeds (of course bridges today generally can and easily do achieve such performance).

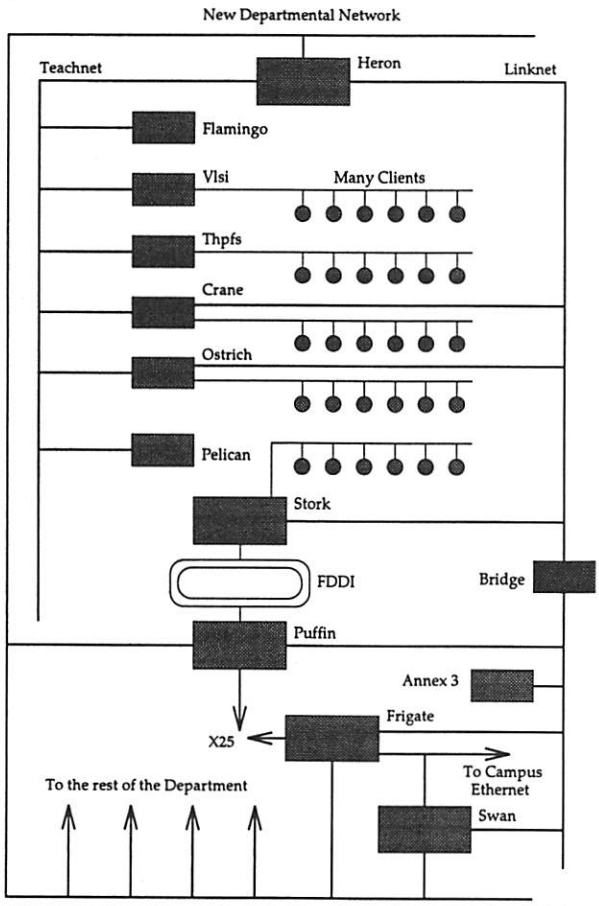


Figure 5: Departmental network overview (cica 1991)

Finally in 1991 the *Gould* came to the end of its life, chiefly due to reliability problems, but the main lesson learnt from it was to avoid at all cost designing any part of a system with a possibly devastating single point of failure, since not only would failures be a problem, but also routine work like software or hardware upgrades too. The *Gould* was replaced by several distributed systems, three Sun Sparc Station 2s (*swan*, *heron* and *frigate*) took over most of its work and the old Sun 3/160 file server was upgraded to a Sparc 4/360 and renamed *Puffin*. An experimental FDDI network ran between *Puffin* and *Stork*, but mainly due to hardware costs it never took off as a possible new departmental network backbone. Figure 5 shows the network overview from that time with a second departmental Ethernet installed to help ease some of the backbone congestion problems seen at that time.

In summary, by this stage multi-homed dedicated servers had been shown to be a good idea, especially where the hardware was capable of easily sustaining the I/O rates required. Single points of devastating failure needed to be avoided wherever possible, hence the distributed approach was much better for most services (e.g., spreading home directory and replicated /usr type file systems, mail, external communications, adequate network routing with alternative routes, etc.). However it has to be recognized that there are additional system management overheads in terms of keeping everything consistent across multiple servers and platforms, but it is possible and tools do exist to help (e.g., *rdist* and *track*).

Meanwhile everyone was buying cars, sorry workstations, bigger faster workstations with higher performance Ethernet interfaces. As a result more and more locations were being networked, the backbone networks were becoming congested carrying an ever increasing amount of traffic, and network cables just seemed to mushroom everywhere. A good few miles of thick and thin Ethernet cable typically ran from the computer room and up the building risers, even filling them to capacity in places, and then off along the corridors to various rooms. Of course physical cable navigation was just as skilled as map reading (where there were maps) and identification signs just as rare and accurate as old road signs at remote country road junctions, e.g., two roads/cables going in different directions to the same place! (and quite correctly labeled when installed). Things change, just as roads get bypassed so do network cables, and just to make things worse there are all those thick Ethernet drop cables too, and the one you have to trace always seems to go for miles crossing several others in its path until you take the wrong turn and follow the wrong one – really just like *twisting country lanes*.

The Problem – Growing Pains

From now on let us mainly concentrate on the teaching side of the departmental network, since it is far more interesting! Having established the idea of

dedicated Network Support Nodes (NSNs) to look after groups of workstations, whilst the NSNs were themselves all well connected to both teaching backbones for good network access, e.g., to all home directories and central services like mail and news, now was the time to expand this successful idea even further (Summer 1991).

Two additional Sun Sparc Station 2 file servers (SS2s) allowed us to significantly improve student NFS file serving by spreading student home directories over three instead of one file server (*Heron*, *Toucan* and *Lorikeet*), all transmitting data to and from clients via the existing two teaching backbones (teach and link net, Figure 6). *Heron* also acted as a second route to the main departmental network. In addition, the two new file servers were also directly connected to a mixture of nine Sun mono ELC and ten color IPX workstations, all with local 207MB

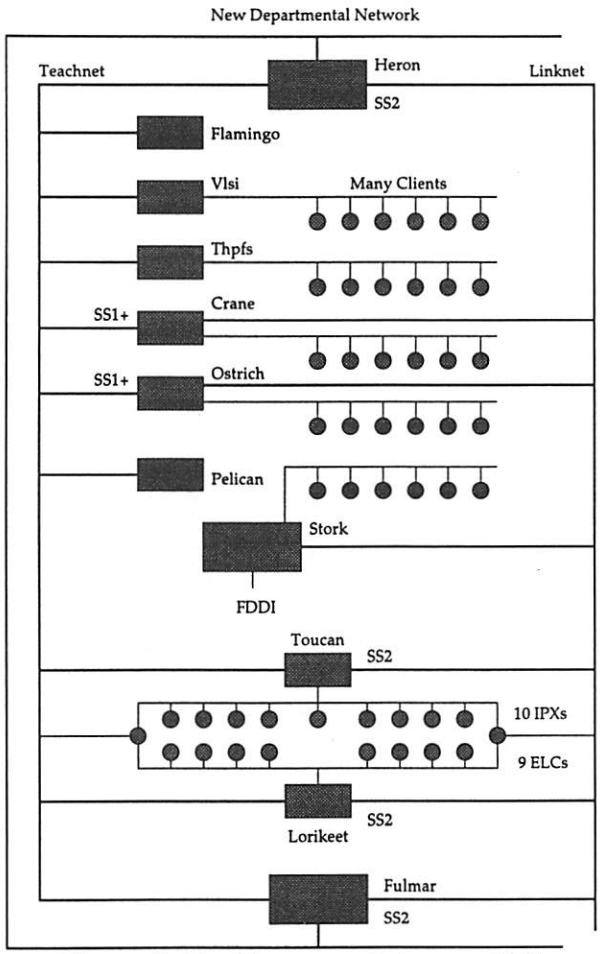


Figure 6: Teaching network Autumn 1991

disks. Just like the first NSNs and their disk less clients, these SS2s locally provided everything their workstations might need and couldn't be stored on the workstation's somewhat limited capacity local disk. However, for student home directory requests, one third would be available locally whilst the remainder would have to come from one of the other two file servers, which were never more than one network hop away from the client workstation. As backup routers, a pair of IPXs also had additional network connectivity to allow the service to be reconfigured should the need ever arise. All in all this solution worked well.

Over the next couple of years another ten IPXs were added with bigger local disks and more memory, but with no additional networking capacity the network soon started to show signs of strain. High levels of collision rates returned, and overall the system was approaching its design capacity. Meanwhile the number of Sun 3s now being used as X-terminals also steadily increased, adding to both network and workstation CPU burdens. Further expansion in the form of three additional Sun Sparc Station 10s (SS10s), two as central CPU servers (*Finch* and *Motmot*) to help improve X-terminal response and one as an upgrade for file server *Heron*, soon took the network at times to near breaking point. Also by then many of the workstations and servers were well under configured for the teaching load now being imposed on them, and so the quality of service degraded, especially at peak times. Not surprisingly users and support staff were increasingly less than happy with the system, especially with the obviously overloaded networks, but not all fully understood why.

Now was the time to study the problem and find a cost effective solution, since further expansion was called for and clearly the existing networking structure could no longer cope.

MultiLane Ethernet SuperHighways

But Why Ethernet?

Early on in the design stage of this project it was recognized that the only viable solution to delivering networking to the desktop was to remain, for now, with Ethernet technology. Quite simply many of the older workstations and X-terminals couldn't accept anything else, whereas for those newer ones with expansion slots available, the costs involved in equipping whole teaching labs with faster interface cards (be it FDDI or its copper based equivalent CDDI, or even Fast Ethernet) was prohibitive and also of questionable benefit considering the overall power of the systems involved. However, any new physical network wiring to the desktop is now installed and fully tested to 100 Mbps specifications, i.e., UTP category 5, making much of the cabling system ready for faster networking whenever it does arrive, be it either of the Fast Ethernet standards, CDDI or even ATM.

Furthermore, there was no perceived need nor support for general workstation networking faster than 10 Mbps, we just needed to get the existing technology working well. Another big plus for continuing with Ethernet was the assimilation of

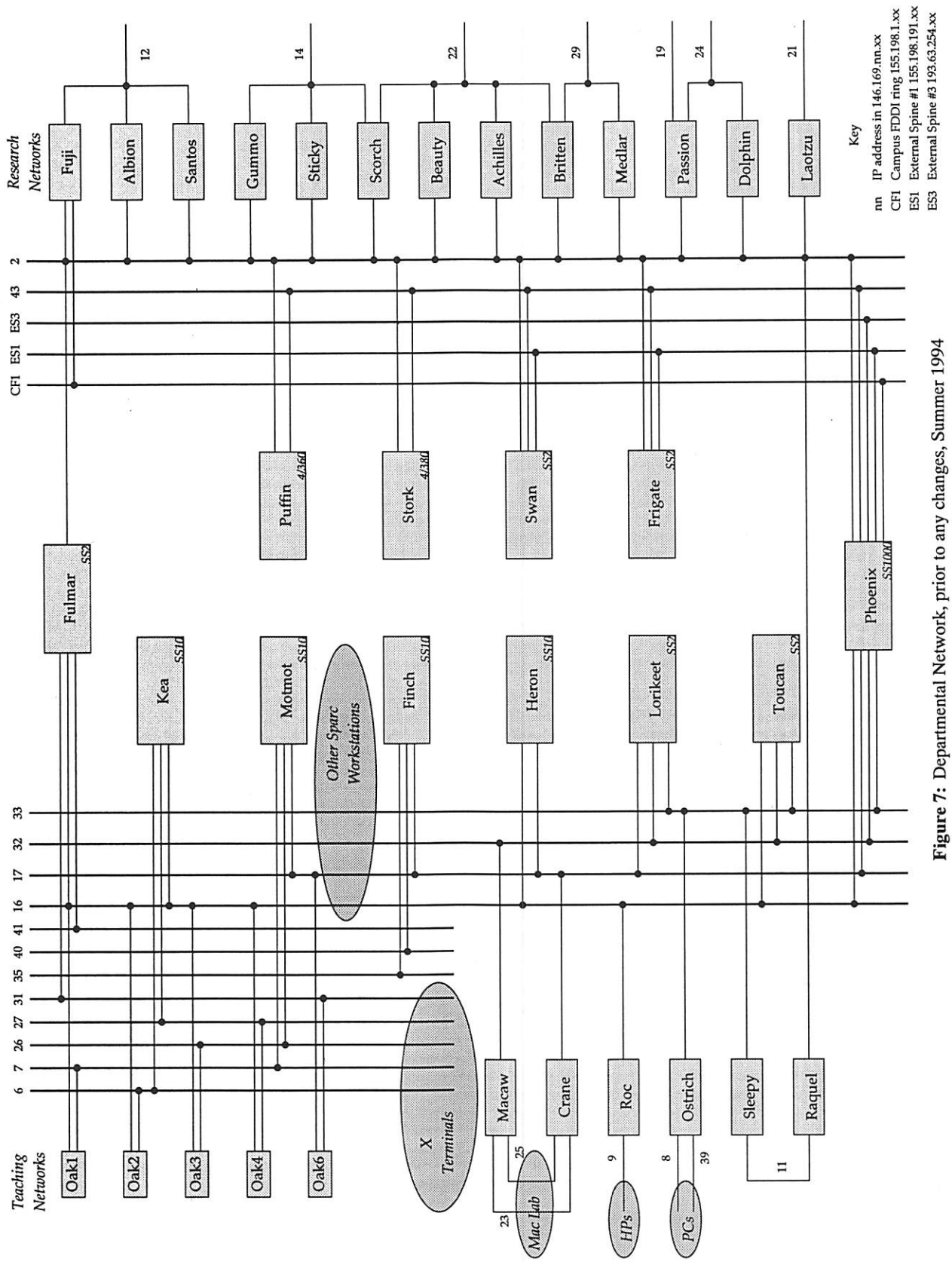


Figure 7: Departmental Network, prior to any changes, Summer 1994

several years practical experience – a considerable asset, especially in terms of rapid problem resolution, traffic capacity planning, and overall understanding – that feel good factor. On the other hand, the extensive deployment to every desktop of some new networking technology, even if previously used in a backbone environment, and however good, carried with it a much higher risk of the unknown – we preferred to minimize such risks.

The Problem Revisited

Having accepted that we would stay with Ethernet to the desktop, the next big issue was how to connect the file servers to their client workstations.

Previously the network had been organized in such a way that no server was ever more than one network hop away from any client. Initially this seemed to be an acceptable compromise between the number of direct network connections (chiefly limited by the number of suitable host server expansion slots), and backbone connections (bandwidth). It was certainly a vast improvement over earlier network topographies.

Originally there was just one coaxial teaching network backbone, extended (by those that knew better) to also support some workstations elsewhere, cost effective perhaps, but when it broke remotely one day, the whole of teaching stopped. Of course one might say that nowadays with UTP wiring and the hubs used now this wouldn't be a problem – but hubs do fail, UTP cables get damaged, the wrong interface gets connected to the wrong hub – things can and do go wrong. So this dual backbone approach remains with us to this day – belt and braces perhaps, but the extra resilience has proved itself time and again to be very well worth having.

In many respects the no more than one hop approach was actually quite good since it also allowed us to successfully implement the idea of distributing class file serving, spreading each teaching class over multiple file servers rather than confining them to a single specific host. Although one might now consider such an approach to have rather obvious advantages and to be the only cost effective scalable approach, resistance stemmed from two areas. The first was financial, where a single new server had been funded for a specific class, and the second was reliability. Not so long ago computer hardware wasn't nearly as reliable as it is today, and so it was felt only fair that should a fault occur the whole class should suffer equally, otherwise some students might have an unfair advantage when it came to marking over others. Fortunately we were able to happily resolve both issues and reap the obvious advantages with few long term difficulties.

However, the one hop approach doesn't scale well with an increasing number of users or parallel teaching (serving two classes at once instead of just one), since as the number of servers and workstations increased, fewer and fewer users found themselves sitting in front of a workstation with direct access to their home directory file server. Furthermore, the number and power of workstations always tended to increase faster than the power of the server(s) assigned to support them. So as the servers became even busier, the latency through them rose significantly, such that even a hop count of one, was one hop too many.

What was generally happening at a user's workstation was that it would try to route a NFS request over the local Ethernet via a busy locally attached file server, which would eventually route it out over one of the two busy busy backbones to the desired file server, which would then reply, or at least try to.

Meanwhile, back at the workstation, things would be going rather slowly, retransmission of UDP packets would be sent out which would again have to be handled by the servers, so increasing network traffic and collisions along with server load. The poor users simply received a worse response and they tended to load balance the system, hopefully coming back later. This wasn't good since full use of facilities wasn't possible nor could demand be adequately satisfied.

So far it would appear that all our problems were chiefly network related, but in fact the workstations themselves were a major contributor to the problem since they were under configured for the tasks now being performed. The most glaring problem was inadequate local disk space and physical memory, resulting in increased network traffic to and from the servers since frequently required pages were flushed rather than remaining locally cached. Workstation configurations would also need to be improved.

There was also a requirement to expand the number of workstations being supported and improve the performance of both the CPU and file servers. Better access to the CPU servers from a large number of X-terminals (based on old Sun 3s) also required urgent attention.

Alternative Solutions

The most obvious solution to improving network performance would be to install a significantly faster backbone, say FDDI/CDDI, or Fast Ethernet or just maybe early ATM. Staying with Ethernet speeds but using an Ethernet switch was also considered, as was the need for file server independent routing, e.g., an additional direct connection of each workstation subnet to a network hub. We also needed to improve the ratio of the number of workstations per Ethernet, i.e., have less workstations per network, which along with the installation of new workstations would require significantly more Ethernet subnets be connected with good network access to the servers.

All this was possible, but so far most solutions also required an expensive network hub, and that required money that was difficult to find. Although the entry price didn't seem too high, by the time one had included all the necessary interfaces for the number of Ethernet networks desired to adequately support all the workstations, they all looked very expensive solutions indeed.

Overall it was preferred to spend funds on workstations and servers, along with more disk space, memory, etc. rather than on an albeit a very high performance network hub, yet still find a network solution to allow good use of the facilities purchased.

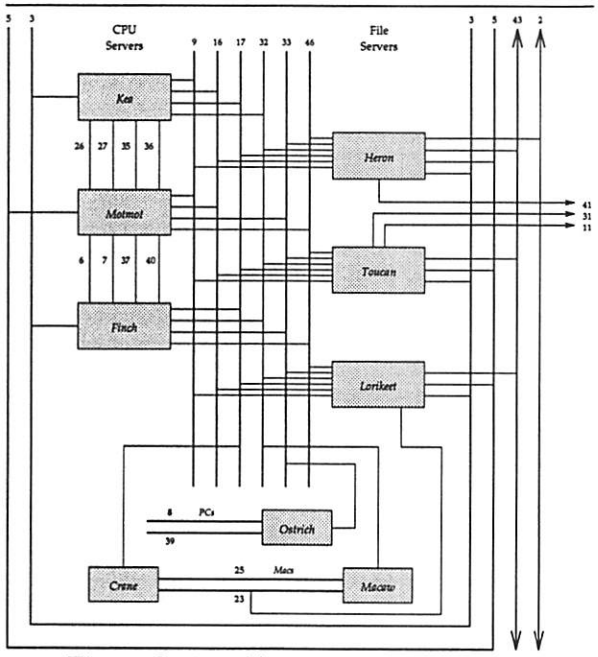


Figure 8: Teaching network Spring 1995

An Ethernet SuperHighway

One promising solution arose from the the idea of significantly increasing the number of workstation networks and directly connecting each network to every file server. This was now possible by upgrading the file servers from old Sun SS2s to SS10s, which were much faster and had four instead of three Sbus slots, and utilizing Sun quad buffered Ethernet Sbus cards which provide four UTP Ethernet interfaces per Sbus slot. Typically each file server would now have two or three quad Ethernet cards plus possibly a combined fast SCSI and buffered Ethernet Sbus card, giving the server two fast SCSI buses (for disk and tape drives) and up to 14 UTP Ethernet interfaces. A dozen of these networks could then be used for workstation subnets since two were still retained for backbone connectivity. Currently each file server only has two quad cards installed, giving a total of around 10 UTP connections, and the remaining Sbus slot remains free for future expansion.

Having such a large number of workstation networks available allowed for a significant reduction in the number workstations per network, down to around eight, dramatically improving the available bandwidth per workstation and helping to reduce the previous excessive levels of network collisions.

The number and power of student workstations also increased dramatically. The main teaching laboratory now supports 18 new 48 MB Sun SS5 color workstations with ½ GB local disks and eight new 64 MB HP 712/80 color workstations with 1 GB drives, along with two 64 MB SS20-SX multi media workstations. In addition ten of the newer IPXs were upgraded to 52 MB of memory and nine ELCs to 24 MB, whilst the remaining ten older IPXs were redeployed for research student use elsewhere.

The direct network access idea has also been extended to supporting X-terminals by placing their networks between pairs of powerful 128 MB Sun Sparc CPU servers *Kea*, *Motmot*, and *Finch* (two SS20s and one SS10), also using quad cards to increase network connectivity and provide direct access to all the file servers and many of the workstations too. Two quad cards are typically installed in each CPU server and eight dedicated X-terminal networks have been constructed. This solves the two old problems of having too many X-terminals per network (now down to an average of 16 mono or 8 color per network), and poor network access to both workstations and dedicated CPU servers.

Twisted Networks

Quite obviously with this amount of networking there is a corresponding large volume of network wiring. Fortunately this is now all UTP which is much easier to handle and much quicker to reconfigure and install, especially in bulk as structured wiring. The workstation end is conventional enough UTP wiring not to merit comment, except that purely on cost grounds small UTP-to-Thin converters are used to wire benches of old Sun 3s where conversion to UTP couldn't be cost justified.

The server end is much more interesting. Typically servers are installed on shelves inside 19 racks, with locally constructed SCSI disk trays on either side. Since there is a very high demand for UTP connections, bulk UTP wiring is run under the computer room floor from the central hub area to each server rack, where it is terminated at a 110-block. From there manufactured UTP patch cables are cut in two and the cut end punched down on the 110-block, providing a fully compliant (and tested) Category 5 cabling system that terminates in a standard RJ-45 which can then be plugged into the server. The other end is just as easy, but instead of a server there are banks of either SNMP managed hubs for the key networks, or cheaper unmanaged ones for less critical uses, e.g., X-terminal networks.

The whole installation was completed over several months, a couple of them very busy involving quite a complex set of phased network changes to allow new networks and services to be gradually phased in whilst others were smoothly removed to be redeployed later. About the only key software aspect worth serious note is that in order to make sensible use of such a highway, lane discipline is very important. DNS needs to return the local IP address of the name requested from the point of view of the local workstation asking, otherwise needless IP routing can and will take place. Also some non-UNIX software can't handle the concept of a host having a dozen or more IP interfaces, shame, but we generally created an alias.

The Results

Quite simply it worked phenomenally well, first time, no problems, good old Ethernet! In fact very few people actually realized what had been done, and apart from a few students who studied the host tables and didn't believe them at first, there have been very few comments. There hasn't been one complaint about network response attributable to the local teaching networks, and it was a cost effective solution delivered on time and within budget.

The Future

The original design has room for further expansion both in terms of supporting more workstations (file serving) and X-terminals (CPU serving). Plans are currently underway for a new student project laboratory which will hopefully integrate well with the existing facilities. Fast Ethernet could also be a very interesting hot topic, especially since the two competing standards have done a lot to bring this technology quickly to market as a working deliverable product. Currently hub prices continue to fall and interface cards are readily available on many platforms, and it will happily run over our existing networking infrastructure, so just plug-and-play. Meanwhile ATM slowly moves through various committees, maybe one day.

Conclusions

This Ethernet SuperHighway approach quickly provided an expandable, cost effective, highly integrated, fast, low congestion and latency, direct (zero hop) connection for each and every workstation to all the teaching file servers. It also directly connects X-terminals between powerful CPU servers, which themselves have multiple direct connections to all the file servers as well. In addition the design is also reasonably network fault tolerant and damage limiting in terms of what becomes unavailable should any single component fail.

It has worked very well and is indeed a very simple solution. Best of all it has many happy users, and room for further expansion to hopefully keep them that way. All in all it has been one of those great behind the scenes successes.

References

[SunUG'91] Stuart McRobert, Divide and Conquer, *README*, Sun User Group, Vol. 6, No. 3, Fall 1991; also in the Sun User Group Conference Proceedings, Atlanta, GA, June 1991.

W. Richard Stevens, TCP/IP Illustrated, Volumes 1 and 2, 1994, 1995, Addison Wesley.

Author Information

Stuart McRobert received his BSc and College prize in Physics at Imperial College London in 1982, where he is now Head of Systems and Chair of Netman (the local Network Management team) in the Department of Computing there. His work has moved from the support of individual systems of the PDP/VAX era, through several local networking firsts (Ethernet, FDDI, UTP wiring) along with the introduction and management of client/server computing and overseeing its subsequent growth into the highly distributed multiprocessor systems of today. He has also been involved in the installation of large parallel systems including a Fujitsu AP1000, and along with a colleague, in their spare time, manages *Sun SITE Northern Europe,* one of the larger and rapidly expanding archives on the Internet, which will be extensively involved in next years *Internet 1996 World Exposition.*

He can be reached by post at the Department of Computing, Imperial College, 180 Queen's Gate, London, UK, SW7 2BZ, or preferably via email to sm@doc.ic.ac.uk.

From Thinnet to 10base-T, From Sys Admin to Network Manager

Arnold de Leon – Synopsys, Inc.

ABSTRACT

Once you have more than one computer at a site, a network is usually required. Often the system administrator also becomes responsible for the network. This becomes yet another task on an already full schedule. To be a successful system administrator, you need a working network; therefore, you now have to become the network manager with a limited amount of time. I will go through the techniques and strategies, including examples from the implementation used at Synopsys, for supporting a network in your spare time.

Introduction

In an ideal world, every organization would have a network manager who has enough time, staff and information to plan network growth and changes. In reality, system administrators are often called upon to act as the network managers for their sites. This paper is about techniques for making that transition.

There are no magic bullets in this paper. It is not about a single tool that will solve your problems. Instead, I hope to present ideas that can help you survive your tenure as the keeper of your network. Here I present the approach that I took and the lessons I learned in designing, growing and supporting the Synopsys network.

The examples herein are from the Synopsys campus in Mountain View, during the deployment of our 10base-T and 10base-FL network and later FDDI/CDDI. I will go through the architecture of the network, including decisions on avoiding the bleeding edge of technology while providing an affordable high-speed network.

I have also included a glossary of working definitions. The definitions are not complete or necessarily even fully accurate. Hopefully they are sufficient to get through this paper.

Background

Synopsys manufactures software and hardware tools for the EDA (Electronic Design Automation) industry. Our products are used by engineers in the design and implementation of chips and systems.

The majority of our computing activity is split among software engineers developing code and users of business applications (management, finance, marketing, etc.). In 1990, when I joined the company, the network was primarily used by developers. Today, all aspects of the business depend on the reliable functioning of the network. NCS, the Network and Computing Services department, is responsible for the network as well as the rest of the computing equipment deployed throughout Synopsys.

In this paper, when I say "we" I am referring to NCS, unless otherwise noted.

Five years ago, the primary Synopsys network was an Ethernet network composed of several pieces of thinnet coax connected by a pair of repeaters (see Figure 1). There were 60 hosts all located on a single floor of a building. The nodes were Unix workstations used in the development of product. There were no routers or bridges in this network. A separate network of 50 Macintoshes also existed. The IP addresses were from the infamous network 192.9.200 (sun-ether). NCS had two Unix system administrators, 1.5 Macintosh administrators and a manager.

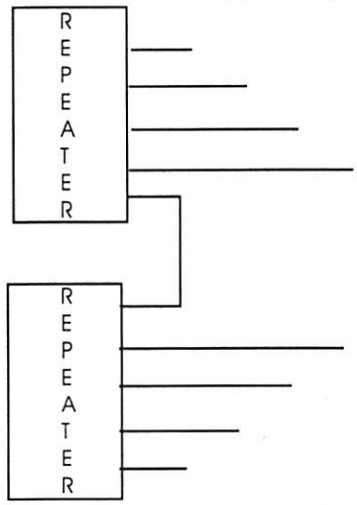


Figure 1: The Synopsys network in 1990: two repeaters and a bunch of thinnet

Today, there are more than 2,500 hosts in a network that reaches from Asia to Europe. The Mountain View campus includes three buildings that share a computer room. There are at least seven FDDI networks, many routers, more than a dozen Ethernet switches and at least 70 10base-T hubs. We have a registered class B network with more than 100 active subnets (more than 30 in the Moun-

tain View campus). We have a 10 Mbit/s connection to the Internet.

The focus of this paper is our campus LAN. I will begin with the general strategy that we used in approaching the problems. Then I will start from the wires and lead into the design of our workgroup and backbone networks.

The challenge

Everyone wants a network that is:

- reliable
- affordable
- high-performance

Synopsys is no exception. We also need a network that can keep up with the rapid company growth, both in terms of the number of users and the capacity per user.

Even though Synopsys was already a successful company, it had a limited staff to deal with the network. As the second Unix system administrator I also became the network manager. I suspect this case is not unusual.

System administrators face enough tasks to keep them very busy. NCS has the added problem of a growing company: we are adding new machines on top of maintaining what we have. This leaves very little time for the network. As a result, planning often becomes an early casualty. Formerly, I expected that once I had a working network I would be able to measure and fine tune it to keep it running in top form; now I know that's not likely to happen in the near future.

Instead, there will be the need to deal with emergencies like having to move 60 people with minimal downtime while adding 10 new users. Reorganizations and moves will also wreak havoc on a network.

Documentation can easily get neglected, posing a potential problem. You may end up following what the documentation says only to discover later that it was incorrect, wasting time and effort. Knowing that the documentation does not exist will at least save the mistakes due to following the incorrect documentation.

Strategy

We needed to have a strategy to approach these problems.

Make Assumptions

First, several assumptions went into the planning process:

- The network is going to grow.
- The network is going to get shuffled.
- Technology is going to change.
- The staffing will eventually catch up.

I discovered that:

- The network grew and is still growing.
- The network got shuffled and is still getting shuffled.
- Technology changed and is still changing.
- The staffing may eventually catch up.

For Synopsys, "the network is going to grow" means more users and more network bandwidth per user. Estimating how much the network is going to grow is important. As a system administrator you should understand how your company is doing. If you can get projections on the growth then you're lucky; otherwise, you will have to guess.

Once we had our assumptions and the problem defined we wanted the solutions to last as long as possible.

Use Standards and Models

Synopsys adopted a strategy of using standards. This idea is not exactly revolutionary. Using industry standards gives us flexibility, which in the short term sometimes appears more costly, but in the long term is generally a better value. Technology moves rapidly, so it is easy to get trapped. Standards allow us to migrate instead of having to do cutover changes.

NCS created standard models for the computing equipment, which have proven to be even more valuable. These models incorporate industry standards as appropriate, but use our own if needed. This means we have specified what kind of equipment is required for most situations. Once a model is in place the effort that went into specifying it can be recouped over and over again with little demand on the network manager's scarce time.

The resulting consistency also allows us to reduce the complexity of maintaining or expanding current configurations. The network management staff can apply knowledge from one installation to make subsequent ones easier. Later in this paper, you will find examples of the standards and models we used as we built our network.

Design Conservatively

Designing conservatively means avoiding being on the edge. If you are not close to capacity or other design limitations you do not have to spend as much time determining if you are about to go over the edge. For example, we try to size things large enough so that we do not have to spend a lot of time measuring and tuning. This may actually mean spending more money on equipment in order to reduce administrative complexity.

Make It Easy

Even the best network will require some work. We set out to make things as easy to support as possible.

Make things easy on yourself and others. To paraphrase [Wall 90], one of the great virtues of a system administrator is laziness. This meant trying to make things easy to use and maintain. If

something was too complicated it would either be bypassed or be done wrong. Being a member of the lazy club, I also sought ways to make it possible for other folks to do the work. This usually meant, once again, making things easy to support.

Learn To Do Nothing

Another important survival technique is doing nothing.

Some of the glitziest items on your network wish list may be difficult or time-consuming to implement. Sometimes the best thing to do is to ignore them. In our case this meant for a long time we did not have a functional network management station (NMS); our staffing was just not sufficient to support one. We have no fancy network maps to this date; our network administrators can draw one from scratch as needed because of the simple architecture of our network.

Look For Value

An underlying principle is the search for value. Implementing the preceding principles often requires spending additional dollars. While we often made the tradeoff to spend money now to save money later, we did not ignore opportunities also save money during initial installation or procurement.

We learned that equipment can be cheap relative to other things. We could have spent money on engineers who would have had to wait for network connections. The "extra" equipment allowed us to accommodate growth and reorganizations with minimal down time. During moves we have been able to bring up the network at the new location before taking down the old network.

Principles in practice

Now we can see how it all went into practice as Synopsys left its old thinnet network for an all new 10base-T one.

Wires

First, we need wires for a network. The opportunity to the use the principles on a new wiring scheme came when we moved to a new building.

General Wiring Design

Our wiring is hierarchical. Nodes are leaves to the workgroup network, a workgroup network is a leaf on our backbone network. This structure makes it easier to debug. When debugging a problem you can just work your way up the tree. Having a simple hierarchy is valuable. At one point, I started creating configurations that had webs of dependencies instead of a simple hierarchy. The result was not a fun time; debugging sometimes became an involved process. A problem in a given area could be caused by what seemed to be an unrelated area of the building or even campus. We have since switched back to simple hierarchies.

The resulting wiring should be easy to trace. I had a limited time to spend on the network. I thought about documenting every cable in network, when I naively thought that documentation was the answer to all problems. I actually tried going through the steps it would take to create and maintain such documentation. I quickly realized that this was not going to be practical with the amount of time and resources available. What was important was to make the system require a minimal amount of documentation. It should be possible to work most problems without having to track down a lot of documentation.

Users being moved cause premature graying in our network managers. We planned for these changes by supporting "any net anywhere." This design meant that any port on the network can be connected to any of the campus LANs. This strategy is necessary to make moves easier to support.

Wiring Our New Building

We defined what we wanted at every station drop. Synopsys had many Macintoshes on PhoneNET so we decided that we needed a PhoneNET port. 10base-T had just become a standard, and we decided that it was going to be the wave of the future. Since 10base-T is a point-to-point system, we provided two ports at every station drop to give us flexibility. The facilities department wanted two phone jacks in each office.

Version 1 of our network jack contained three cables, each with four twisted pairs (see Figure 2).

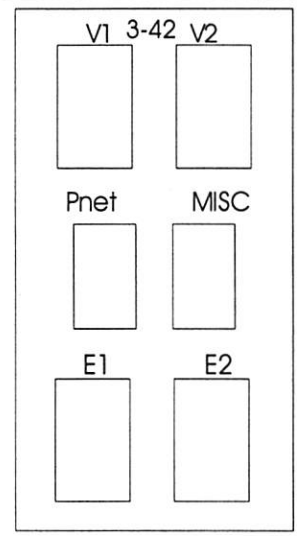


Figure 2: Original ports per station drop, ports in today's station drop

Cable 1 was dedicated to voice. The pairs were split between two jacks – three pairs in one and one pair in the other. The three pairs in one port allowed it to be used for digital phones. Only one pair is required for an analog phone.

Cable 2 was assigned for PhoneNET. Since PhoneNET needed only one pair in an RJ11 jack, we decided to connect the remaining pairs to a second RJ11 for "misc" use, such as a serial device.

Cable 3 was for Ethernet. On our vendor's advice we decided to use a new high-grade cable for this one. This cable would later be classified as Category 5. The wires were split evenly between two RJ45 jacks to support two 10base-T ports.

We chose configurations that we believed to be flexible. We had some pretty good success. The wiring based on those standards, over four years ago, is still in use today. I learned a great deal from the experience.

Those insights have been applied to what we do today. Our new wiring is much simpler (see Figure 3). The voice part is unchanged. Instead of having four different data jacks with different wiring in them we now use two identically wired RJ45 jacks for every drop. A good specification to follow today is the EIA/TIA-568 standard.

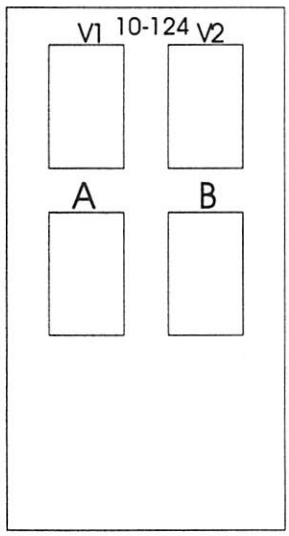


Figure 3: The current version of our network drop.

This configuration is quite flexible. It can be used without changes for:

- RS-232
- PhoneNET
- 10base-T
- telephones (analog and digital)

Many of the new technologies require Category 5-rated systems so most new wiring installations being installed today are being built to Category 5 specifications (see the EIA/TIA-568 standards). The resulting increase in demand has driven the cost of a Category 5 system lower and lower. It just does not make sense to do any less today. And if you do get a Category 5 installation you will be able to support any of the following newer technologies:

- CDDI
- ATM
- 100base-T
- 100base-VG

We learned to avoid overly specialized wiring. Our initial install had a jack dedicated to PhoneNET. The value of those ports is diminishing rapidly as we move our Macintoshes to Ethernet.

Another thing we learned was not to split wires between two jacks. 10base-T only requires two pairs. In our first installation we had a four-pair cable split into two 10base-T jacks. It seemed clever at the time, but we failed to consider what future networking technologies might require. Now, we are regretting that decision as we face the prospect of reterminating those jacks to support emerging technologies. If you must split wires, make sure you leave enough slack in the cable to allow you to fix it in the future.

It is important to provide enough ports per station drop. The two RJ45 jacks allow us to connect two different devices to two different networks without any trouble.

In case two Ethernet ports are not enough we can "split" a fully populated RJ45 into two 10base-T ports using an adapter. And if that is not enough we can install a hub or a fanout in the office.

The station jack is connected to a wiring closet (sometimes referred to as a phone closet, IDF or SDF).

We specify that the cable runs be 90 meters or less. The commonly accepted limit for 10base-T runs is 100 meters. We want to leave room for the patch cables. Also note that a good Category 5 installation can support runs significantly longer that 100 meters for 10base-T. A conservative installation creates a safety margin that can save work down the line.

Our initial install had the wires going to punchdown blocks. The wires were then cross-connected to patch panels. To save money, we only cross-connected one of the Ethernet ports and the PhoneNET port. The "misc" port and second 10base-T port were unconnected. We thought these ports would be rarely used and we could connect them on an as-needed basis. This second Ethernet port is now used far more than we had anticipated, and any money saved initially has long since been spent in the trouble that we go through when enabling the second Ethernet port. We failed to apply the principle of making things all alike to simplify support.

The punchdown blocks were also supposed to provide the flexibility of changing the pair allocations. By changing the cross-connects we would support future technologies. This has not turned out to be useful. We have not needed or wanted to fix the ports this way. Sometimes, predictions go wrong and we just take the lesson and move on.

We now have the station cables terminated onto panels directly. This reduces the number of connections that need to be made. This saves on installation costs and the higher-speed technologies require (or at least prefer) fewer interconnects. Instead of worrying about changing how the pairs are split and what pairs to cross-connect, we have all the wires connected.

Make sure that slack is left on the cable. Sometimes even the best designs will need revision. We have a floor that was wired before the Category 5 standards were defined completely. The patch panels may need to be reterminated to be fully compliant with Category 5 at a future time.

You may also need the slack to move things around later. Our first building did not have a wire management system at all. Imagine a wiring closet in use: there will be many patch cords. It will need wire management. We have spent a of lot time and money cleaning up after this oversight. We still have a lot of time and money to spend before we recover completely.

Backbone Cabling

If a building has multiple wiring closets, install "building backbone" wire. You may need fiber to do this. We installed 12 pairs of 62.5 μ multimode fiberoptic cable between the main wiring closet in the building and each of the other wiring closets. When we chose 62.5 μ multimode fiber, we knew we could use it for FDDI and 10base-FL, and today ATM, 100base-FL, etc., also can run on it.

Our wiring closets were close enough so we could also run a copper backbone network from the building closet to each of the secondary closet. We ran 24 jacks. Since fiberoptic cable is significantly more expensive we were able to save money with this installation.

If you cannot run a copper backbone you should consider running more fiber. Look at the size of the area the closet is serving. Keep in mind that we had a design goal of supporting any subnet anywhere so it was possible to have large number of networks in an area. This required that we be able to bring a large number of networks from elsewhere to any wiring closet.

We also took advantage of the copper backbone between the closets to share a terminal server between the closets. We connected the serial consoles of our network devices to the terminal server. The serial consoles on the terminal server allowed us to access the consoles of the devices even when they were off the net. Console servers are discussed in a paper from LISA IV [Fine and Romig 90].

Campus Cabling

Our campus backbone resembles the building backbone, except it was all fiber since we were going between buildings. We have a strong emphasis on star topologies, so we ran enough fiber between the buildings to take all the fibers in the closets back to our computer room. For example, in each building with four closets, we ran 48 pairs of fiber from the building main wiring closet to the campus computer room.

We have not had a need for single-mode fiber yet. I suspect this will be more of an issue if our campus gets larger and we need the longer distances that single mode supports.

Labeling

Labels are critical to make something self-documenting. You can skimp on other documentation but it is painful when you do not label well.

As you go through the trouble to label, make sure the labels are correct. It is better to have something unlabeled rather than incorrectly labeled. Whenever possible use labeling systems that are permanent. Examples of this include station numbers and port numbers. Those kind of labels should be well-attached (permanent). You do not want it to change. Other labels, like the name of the network or machine, should be removable or erasable since they can change. They should be easy enough to remove so you do not get stale labels, but attached securely enough so they do not fall off accidentally.

In our network, each station drop has a number in the form *nn-mmm*, (e.g., 10-102). The first number signifies which wiring closet it is in. We decided to assign a unique number to all our wiring closets in the campus. This means we do not need to know what building a jack is in. We can infer it from the station number. We also make sure that the there is only one number for the jack. The entire set of ports on a given plate are 3-42. We had sub-identifiers, like V1, V2, P-net, E1, A and B, for each port.

Do not take for granted that your vendor will its works correctly. Make sure that you discuss this issue with your wiring vendor. We had our vendor redo labels in a section when they failed to communicate our requirements to their technician. Consistent labels are important in troubleshooting. When there is a problem, we can get the user to read the number and we know which wiring closet to go to.

In the closets we have patch panels and equipment labeled with the subnet/network number that they are on. This makes checking a connection quick and easy. All the information is out where you need it, not hidden away in some book or file.

An important result of good labeling is that it is system administrator friendly. Good design allows our system administrators to do a lot of the first-level debugging. They can quickly check a given station to see if it is connected to the right place. If a user moves to a new office the system

administrator can easily move the connection by following the labels and cables that are in place. They can also just as easily add another connection for a new system.

Patch Cords

You are going to need patch cords. Lots of them, actually. Here are some rules of thumb to make them easier to manage and take up less support time.

1. Avoid making your own. You are probably not good enough! Making good cables takes practice and patience. It is not cost-effective for you to make them. You will probably pay for the cables twice, first in the time it takes to make it and second when you debug a problem caused by your cable. Making good cables requires good equipment. I have wasted a lot of time fighting with a cheap crimping tool.
2. Make sure your patch cords are correct. Qualify your vendor. You will be surprised at the number of vendors that produce incorrect wiring. A well-known workstation manufacturer used to ship an incorrectly wired 10base-T cable. Also, make sure your purchasing department does not switch sources on you. Qualify multiple sources to make the purchasing folks happy. Spot check your cables as they come in. You need to learn enough about cables to know if you have good ones or not. A cable tester, one that can check the quality of the cable, can help provide the needed expertise in checking patch cords and wiring.
3. Get your vendor to serialize your patch cables. It's cheap. Make sure the serial number is on both ends. This will make it easier to identify ends of cables. I initially tried serializing them on my own. Don't even bother. It was time-consuming, boring and the labels fell off. Get your vendor to do it. Our serial numbers are of the form *ttt-lll-nnnn*, where *ttt* is the type, *lll* is the length and *nnnn* is a sequence number. This is particularly handy for long cables since you don't have to follow the cable all the way from end to end to know that you have the right one.
4. Consider color coding patch cords. We use the colors to identify groups (subnets) within a wiring closet. The downside is you will have to stock more cables. If you don't keep the right colors handy, your color coding will become almost useless as the wrong color will be used if it was the only one available.
5. Make sure you keep a good stock of the cables that you need on hand. Keep multiple lengths, store some in the wiring closets. Again they are cheap. An inadequate stock of the right cables will result in all kinds of cables appearing in the wiring. There are many RJ45 cables out there, including ones that use flat untwisted wires. These cables are not suitable for 10base-T. What is even worse is they will often work, but only marginally. You do not want to have to debug that problem.
6. Make sure that it is possible to tell the cables apart at a glance. A crossover replaced with a normal patch cable will not work. If you have CDDI, a CDDI crossover cable will not work in place of a 10base-T cable or a 10base-T crossover. We now get colored boots on the connectors of our cables. These boots serve two functions. First they protect the tabs on the cables so that it is possible to pull a patch cable through a group of wires without getting it caught. And second, they identify the cable type. (Since we used the color of the cable already for a different purpose, we could not use that to identify the type of cable.) Boots of different colors help make it easy to quickly identify the type of cable.

Plan For the Space

When sizing and building a wiring closet make sure that the closet has enough room for the space that it is supporting. It is unlikely that you will get a chance to rebuild your wiring closet and get more space in the future. For example, an area that is sparsely populated today because it is a warehouse may hold a dense population of engineers in the future. The Synopsys warehouse has moved three times over the last four years. The previous locations now have cubicles in them. Make sure that the wiring closet can handle those kinds of changes. You can approximate a maximum port count by taking the square footage and dividing by 100 (a typical office in Synopsys is between 80 and 140 square feet). You will also have to take into account exceptions. For instance, lab space may require a significantly higher density of of network connections.

Planning for space also means making sure that a large conference room has enough drops. Yes, it may seem like a waste but it is just about impossible to know how the room is going to be used in the future. The extra drops will make it easier to convert the room into temporary office space. It also means that it is more likely that the drop will be on the correct wall when you want to set up a demo in that room.

Large executive offices also need extra drops. I will guarantee that if you place only one drop in those offices, the vice president will place their desk at the farthest possible point from the network drop.

Plan For Change

Plan enough capacity to minimize, if not eliminate, the need to make changes.

If you must make changes then make additions instead of modifications to the existing wiring. For example, if you discover that a drop is not where you need one, don't move a nearby one, add another one instead.

The vast majority of our wire problems are caused by additions and changes in the physical plant. It is generally better to have planned to have enough drops to begin with.

If you are using a contractor for wiring, the additions or changes will typically be done by the "most available" technician. Often "most available" is the "least able." This is usually not the same person that did the initial install. So you end up supervising more or correcting errors.

Plan for Growth and Spares

We have two RJ45s per drop. This allows us to support the growing number of machines that users ask us to connect to the network. The extra port is also useful when a problem occurs in the wiring. We can "borrow" the spare port from the cubicle or office next door. This is an important trick to be able to pull; we want to avoid absolute emergencies. We would rather schedule the work at our convenience.

Plan for Future Technology

Try to look at what new technology is being developed. We now use Category 5 wire; 10base-T does not need it, but higher-speed stuff almost certainly will. On the other hand, accept that you can not predict the future perfectly; we already know parts that we got wrong. There are parts of the network that will not support the new stuff without some work. The ports with split 10base-T; for instance, ports will require retermination.

Putting the wires to work

Once we had the physical plant done we needed to actually start using it. We would need to define a backbone and the networks that would feed it.

Collapsed Backbone

Instead of a traditional backbone which might require high-speed media, we used the high-speed backplane of today's routers (and bridges, or "switches"). Our first implementation used a multiport router with all Ethernet ports. An Ethernet backbone could become a bottleneck. It could be FDDI to make it faster, but that would have been expensive. By using a collapsed backbone, we are able to avoid investing early in an FDDI backbone.

Hosts Do Not Route

Hosts generally are lousy routers and routing usually detracts from their performance. Using hosts as routers complicates configurations and the network becomes harder to debug. Simplification is an important win when you have limited time and staff to deal with network problems. Make sure that the machines have the *ip_forward* (or equivalent) kernel variable turned off. Tracking down packet loops in networks is not fun.

Services Should Be Local

Services for a group should be as network-local as possible. We do not want to depend on the performance of routers to provide reasonable access to most services. It is too easy to get caught in the trap of trying to figure out how to deliver "wire speed" from one place to another when the correct answer is to locate the service closer. Some servers may require multiple interfaces to get them "local." You will need to pay attention to how packets get to these hosts [Swartz 94]. This may be cheaper, and may perform better, than having a high-speed device in-between.

Services Network

A group may need to "publish" or provide a service to the rest of the network from one of its servers. In order to minimize the impact of these shared services, I created a "services" network. A group's server that is providing the service will have an interface on this network. Access by "remote" users go through this network and interface. This has an important benefit: non-workgroup-initiated traffic is kept to a minimum on the workgroup backbone interface.

Workgroup Model

First, we defined a workgroup as group of systems and their users that shared resources (typically computer systems). Our workgroup model resulted in the workgroup sharing a network.

Our initial workgroup network was a single Ethernet segment (see Figure 4). It would have 12-36 standard workstations (per our computing model) with a target of 24. As a workgroup grew it would be split into two or more groups as appropriate. This model gave us the ability to project our networking equipment needs.

When we were implementing this network, FDDI was the showcase technology at Interop. Showcase meant that the technology was still expensive and too new. Instead of going for the latest technology, we gave our large shared servers multiple Ethernet interfaces. This approach allowed us to keep servers and clients topologically near each other. The performance of the backbone router was not as critical since it was not a part of the routine traffic between workgroups clients and servers.

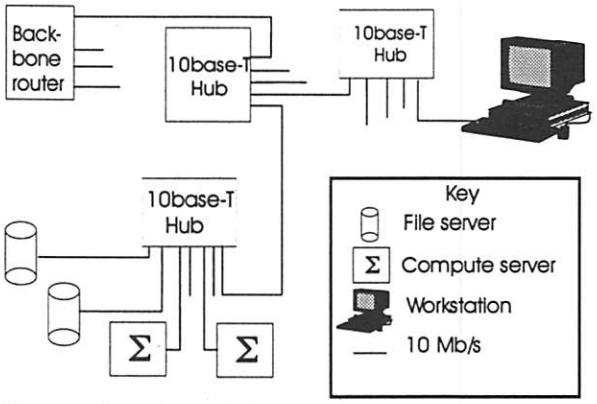


Figure 4: The original workgroup network: One Ethernet connected to the backbone router

As workstations became more powerful it became clear that a single Ethernet segment was not going to be sufficient for a workgroup. It was not practical to split a workgroup network into separate sub-networks that were each smaller than the actual workgroup. Part of the solution is to split the workgroup network into multiple Ethernet segments. Our workgroups today look very similar to the ones we had four years ago. All that we did was modify the root or base of the group. Instead of a hub at the root we now use a high-performance bridge. This modification allowed us to give each group more than a single 10 Mbit/s Ethernet.

The new lower-cost, high-performance bridge/routers (switches) also allowed us to redefine the size of a workgroup. Previously we had to make a workgroup fairly small to fit within a single Ethernet. We are now able to allow a workgroup to be bigger since we can have a fairly large number of Ethernet segments available. This simplifies the administration of the groups. Since there are fewer logical workgroups, there are fewer moves that actually crossed workgroups. This also simplifies some of the resource sharing problems, the larger workgroups can share a server more readily.

A mismatch between the client and server network pipes can occur when switches are deployed. In our computing environment the traffic predominantly is between client and server instead from peer to peer. We have many devices talking to a few servers. It was possible for the clients in one group to generate combined traffic in exceeding the bandwidth of a of single Ethernet. Our solution was to give the servers FDDI interfaces. Note that we were able to delay this decision until FDDI interfaces and hubs had dropped significantly in cost. The introduction of FDDI over twisted pair wires (TP-PMD or CDDI) further reduced the expense required. This led us to implement using workgroup bridges with FDDI interfaces.

There are also additional factors that influenced our workgroup model.

Keep High Speeds in the Computer Room

All along we tried to keep the need for high-speed networking away from the desktop machines because it is much easier to retrofit a computer room for new technology than a campus. We have been able to restrict high speeds to the computer room by keeping the servers in the computer room.

We have taken advantage of our current configuration by providing our compute servers and file servers, located in the computer room, high-speed (CDDI) connections. The higher-speed connections give the compute servers improved access to the file servers without having to rewire the buildings.

At Synopsys, the average desktop machine has not required more than Ethernet. When we started four years ago a workstation shared an Ethernet with about 30 other workstations, and today it is closer to 10. We will eventually be able drop it down to a dedicated Ethernet.

By restricting higher speeds to the computer room we have been able to continue to use the same wiring we had four years ago. In that time, we have significantly increased the available bandwidth between servers, as well as between the servers and the desktop machines, without having to redo our station wiring.

Segregate Traffic

Users' perceptions of network latency are important. The workgroup Ethernet switch architecture also allows us to dedicate a segment to X terminals. The logic behind this approach is that NFS uses large packets and can delay the small packets X usually generates. The effectiveness of this change has not yet been measured.

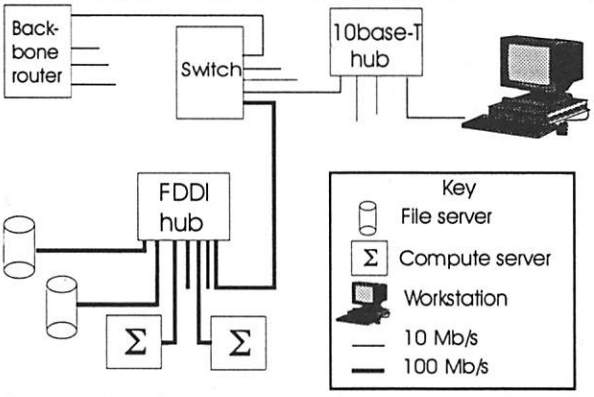


Figure 5: The current workgroup network, built around a local high-speed network and switches

The result is our standard workgroup now consists of a FDDI/CDDI network connected to some number of Ethernets by high-speed bridges/switches (see Figure 5). Compute servers are attached to the the high-speed network (FDDI/CDDI) along with the file servers. An Ethernet segment is intended to be

used by 6-12 standard workstations. An Ethernet segment is reserved for X terminals.

An interesting problem arose in the specification of the equipment for this configuration. Choosing to bridge between Ethernet and FDDI led to an interesting trade-off. I could have chosen a simple, less-expensive device to link the Ethernet to FDDI, with the caveat that it would not handle the different MTUs between Ethernet and FDDI. We would have had to ensure that the different host configurations deal with the different MTUs correctly. Instead we chose to specify the more expensive box that could deal with the problem correctly. This choice allowed us to not to have worry about the MTU problem.

Note that in four years a workgroup has gone from a single 10 Mbit Ethernet segment to several (8 to 24 today) and a local 100 Mbit (FDDI) network. The hub and wiring from four years ago are still in service. Even though the available bandwidth for a workgroup has changed by at least an order of magnitude, we have been able to keep the backbone interface at 10 Mbit. We have not added a higher-speed backbone connection because we do not require one yet. I expect that by the time we need one, the prices will be lower. Continuing to use existing equipment helps keep our costs down.

Manage the technology

Managing the technology is an important component of supporting a network.

Simple Routing

Because of the structure of the network, our routing configuration is fairly simple. This simplicity helps cut down the debugging time. It is extremely rare that we have to debug the network using *traceroute* (8). Our system administrators also know the standard IP configuration for an end node. We have a convention for the IP address of the default router on a subnet. A system administrator can configure a node on almost any of our subnets by just understanding this convention.

Ignore the Hype

We were conservative in that we ignored a lot of the hype about new technology. The leading edge can often be the bleeding edge and new technology usually has higher costs. The vendors and the market have a learning period to go through. First-generation products always have bugs. Performance may not yet meet expectations. Unless you have time to spend helping the vendors make the products into the solutions they are hyped to be, it is best to avoid this time sink.

Stay Tuned In

On the other hand, you will have to make a lot of assumptions. To be effective you do need to listen. You need to guess when it's safe to take on board a new technology. If you are unsure about a technology then you may need to test and prototype the new technology. It is a tough balancing act, avoiding products that are not ready yet versus deploying a new technology that is valuable. We ran FDDI in a test configuration for a long time before adopting it as standard. I kept an eye on trade shows. If a technology is taken for granted and is just used without a lot of hype then it is probably mature technology.

Limit What You do

Just because something is possible does not always make it a good idea. Some well-known routers can route just about anything, but being able to route something doesn't mean that you should. I tried to make it easy on myself and the other system administrators. For example, we defined our backbone protocols. We route only TCP/IP and AppleTalk. This simplifies router configurations. Remember: limited staff, limited time.

Augmenting your staff

Synopsys went from about 100 to 1000 people with only a part-time network manager. I did not have the time to do all the work, so I looked for ways to have other people do the work.

Vendors

Vendors can help you track the market. They may present solutions to problems. By asking them how to solve a problem and having them explain their solution, I was able to learn something new. I did have to remember to be skeptical. I also sometimes just had to take (believe?) what they offered, either because I did not have time to research the topic further or because it was an area where I did not have enough expertise.

You should make sure that vendors document the work they do. Make sure you agree up front what the requirements are. This includes having them test the installation and document the test results. Make sure the agreement is in writing. I have made the mistake of not getting the details documented earlier on. In these cases I had to live with what I got since I did not have time to track it down later.

The Other System Administrators

System administrators can be network administrators. We try to make it possible for the system administrators to do some of the work. We choose to give system administrators access to wiring closets. This means they can do a lot of the work normally handled by the networking staff. Because the scheme is simple enough, they can handle adds and most changes. This also requires some tolerance; errors and mistakes will happen. You can decide for your site if it's more trouble that it's worth.

Tuning, growing, maintenance, etc.

Tuning is a tough problem, sometimes you have to guess what you need. You can try to increase the speed of your nodes and your users will tell you the network became worse [Seifert 95]. You may discover peculiar performance characteristics, some machines performing well while others suffer [Molle 94]. Ethernet has been around long enough to be a mature technology, yet unexpected problems still crop up. Fortunately, there are people looking at the problem. You may have to wait for someone else to solve the problem for you.

One of the reasons for designing the network conservatively is to avoid these problems. By having enough capacity you can ignore tuning for long stretches of time. You will also hopefully avoid triggering some of the bugs that appear at the edge of the performance curves by having this extra capacity.

You should also design conservatively so that you do not have to track down all the details. We avoid pushing things to the edge so we do not have to sweat the details. Examples include keeping wiring short so we do not have worry about a station being outside the design limit if we use a slightly long cord. Another example is our Ethernet repeater configuration. We designed it to be shallow. Ethernet has a limitation on how many repeaters can be between two stations. By staying away from margin we know that if we install an extra hub (repeater) at the edge that it will not push us over the limit.

Metrics are hard. Our management wanted a single number to reflect the health of the network. I deferred this problem. See [Hogan 95] for an approach NCS took at trying to address this problem.

We want minimal maintenance. Equipment is cheap, but cheap equipment is expensive. Reliability should be one of your purchasing criteria. Keep spares on site for anything that you can't live without. This seems obvious but it is easy to forget.

Do not forget that standardized parts yield in spares for "critical" parts by stealing them from "less critical" areas. Standardization also helps in stocking spares, since you have fewer spares to stock. Finally, standardization lowers your support costs: once you know one, you know the rest.

Another use for a spares inventory is to deal with unexpected growth. By keeping a stock of parts we can install in a timely manner and deal with a surprise move or acquisition.

Make sure network parts are reliable. Where appropriate, use equipment designed for the job. This may mean using things like redundant power supplies. You do not want to be paged because of simple failure. Save the heroics for the real hard stuff.

Miscellaneous

Don't be afraid to guess. We had to guess. There is just not enough time to get all the details. Chances are you are the expert for your site. You know how systems are used and can design the network to support it.

There are advantages to being both the system administrator and the network administrator. Instead of choosing a path that was just optimal for network administration we chose one that was good for the group. When one our of workgroups moves, the machines do not have to be reconfigured. We make it easier on the system administrator by making the same network available at the old and new locations. This also reduces the downtime for the users involved. It requires a little more preparation and equipment on the networking side but it saves work for the system administrators.

Radios are step savers. Consider having at least a couple of walkie-talkies available. Two people can often debug a network problem a lot quicker than one. We have vendors who do work for us who end up borrowing our radios.

I am a big fan of blinking lights. I think vendors do not provide enough of them. Our standard transceiver has power, link, transmit, and receive lights. The hubs have link lights. Our system administrators can easily determine if a station is on the network or not. Our help desk can also quickly determine if a user has basic connectivity without having to walk to the user's office. We are lazy, after all.

Make things easy on yourself. We made our AppleTalk network number be the same as the IP subnet number. This makes debugging easier for us.

Network management

Network management stations (NMS) are thought to help by giving advance warning of when upgrades or expansion or repair is needed but they are are not panaceas. Deploying and supporting an NMS is not a simple undertaking; additional resources are usually required to configure and maintain it. Even if one managed to get one deployed and supported, there may still be no one to dispatch to solve the problems that it, the NMS, reports.

I tried deploying an NMS three years ago, and it didn't work. The problem is that an NMS requires time to support and maintain and I did not have the time. We did, however, continue to plan for one by specifying and purchasing equipment that supports SNMP.

We are deploying a new one now. Management decided that this is important. We have more staff now so maybe it will work and our networking hardware was ready to support it. We will see how it goes.

Conclusion

The network is alive. We have somehow managed with a limited staff (mostly me) taking care of it. Our design has been surprisingly resilient in the face of all the demands and changes. Most importantly, my successor has not killed me yet.

We now have two full time folks supporting our network. The problems strangely haven't changed too much. Synopsys continues to grow so our challenges continue.

Acknowledgments

I would like to thank my merry band of proofreaders: my wife Laura, Dave Stuit (who can tell the difference between an em-dash and an en-dash), Jeff Jensen (who took over running the Synopsys network), Becky (who let her husband, Jeff, take over running the Synopsys network), Ted Lilley (who recently joined Jeff in running the network), Dave Clark and Christine Hogan. Additional thanks to Erin Elder and Michael Jensen, who had the misfortune of being at the wrong place at the wrong time. They all tried valiantly to find all of my typos, missing words and poor writing; any remaining errors are my fault.

Geraldine de Leon created the figures based on my crude sketches.

Also thanks to Paul Evans for encouraging me in this process, Eric Berglund for making this paper my project, and finally, to Randy Collins for letting me build the Synopsys network.

Author Information

Arnold de Leon is a senior system administrator at Synopsys, Inc. Arnold is the current president of BayLISA. He was also a founding board member of SAGE. He has a bachelor's degree in mathematics from Harvey Mudd College. Reach him via U.S. Mail at Synopsys, Inc.; Building C; 700 East Middlefield Road; Mountain View, CA 94043-4033. Reach him electronically at <arnold@synopsys.com>.

References

[Wall and Schwartz 90] Wall, Larry and Randal L. Schwartz, *Programming Perl*, Sebastopol, CA: O'Reilly & Associates, Inc., 1990, p. xviii.

[Fine and Romig 90] Fine, Thomas and Steve Romig, "A Console Server," *Proceedings of the Fourth Large Installation Systems Administration Conference*, The Usenix Association, 1990.

[Swartz 92] Swartz, Karl L., "Optimal Routing of IP Packets to Multi-Homed Servers," *Proceedings of the Sixth Large Installation Systems Administration Conference*, The Usenix Association, 1992.

[Molle 94] Mole, Mart L., *A New Binary Logarithmic Arbitration Method for Ethernet*, Computer Systems Research Institute, University of Toronto, Toronto, Canada, April 1994, available for anonymous ftp from ftp.utexas.edu as /pub/netinfo/ethernet/ethernet-capture/report.ps.

[Seifert 95] Seifert, Rich, *The Effect of Ethernet Behavior on Networks using High-Performance Workstations and Servers*, March 1995, available for anonymous ftp from ftp.utexas.edu as /pub/netinfo/ethernet/techrept13.ps.

[Hogan 95] Hogan, Christine, "Metrics for Managers," *Proceedings of the Ninth Systems Administration Conference (LISA IX)*, The Usenix Association, 1995.

Working Definitions

This is not intended to be a definitive glossary. The definitions will not be perfectly accurate. They are what could be called working definitions, sufficient to get through this paper.

100base-T, roughly, Ethernet sped up 10 times.

100base-VG, a technology for 100 Mbit networking.

10base-FL, a standard of running Ethernet on one pair of fiberoptic cables.

10base-T, a standard for running Ethernet on two pairs of twisted wires.

62.5 μ is a common size for fiberoptic cables. It refers to to the inside core diameter; the cable itself is much bigger.

AppleTalk is a network protocol designed by Apple, commonly used by Macintoshes.

ATM (Asynchronous Transfer Mode), a faster networking technology that is expected to become the standard of the future.

bridge is a device that connects networks together and makes them look like one network. It does not understand the protocols of the packets it forwards. Forwards broadcasts to all ports.

Category 3 is a standard for cables. See **Category 4** and **Category 5**.

Category 4 is better than **Category 3** but not **Category 5**.

Category 5 is a standard for cables; supports "high-speed" networking. The most common cable used for data applications today. Also used to described an entire wiring installation. A Category 5 installation requires Category 5 wire and Category 5 rated equipment. See **TIA-568**.

CDDI, see **TP-DDI**.

compute server, a computer system assigned to run compute-intensive applications. See **file server**.

concentrator, a box that has ports and looks logically like a wire; a multi-port repeater. For example, a 10base-T concentrator connects several 10base-T ports together into one network. Also known as a **hub**.

crossover cable, a cable that changes the signals so that transmit is now receive and and receive is now transmit. A 10base-T crossover is not the same as CDDI crossover. See **straight-through cable**.

drop is short for **station drop**.

EIA-568, see **TIA-568**.

Ethernet is networking technology for connecting devices at 10 Mbit/second.

fanout is a multiport transceiver.

FDDI is a networking standard that runs at 100 Mbps.

file server, network device that offers file service.

FOIRL is the precursor to **10base-FL**.

hub, see **concentrator**.

IDF (Intermediate Distribution Frame), telecommunications speak for a wiring closet.

MTU (maximum transmission unit), the largest packet that can be carried on a network.

multimode fiber, a kind of fiberoptic cable. Usually used for FDDI and 10base-FL.

multiport transceiver allows multiple devices to share a network port.

network administrator, the networking analogue of system administrator.

network manager, another name for network administrator.

NMS (network management station), a big expensive collection of hardware and software that is supposed to make the life of the network administrator easier. Usually requires a lot of work to maintain.

patch cables, a cable used to connect two ports on patch panels.

phone closet, see **wiring closet**.

PhoneNET, an AppleTalk network that was designed to use one pair of wires in a two-pair phone cable.

repeater takes a signal into a port and regenerates (repeats) it to its output ports.

router, a device that connects networks together. It actually understands the part of protocols that it forwards (see bridge).

RJ11, the connector used by phone jacks; it can contain up to to six conductors.

RJ45, a bigger version of the RJ11; it can contain up to eight conductors.

RS-232, a standard for serial communications.

SDF (secondary distribution frame), see **IDF**.

single-mode fiber kind of fiberoptic cable. Allows the use of lasers to support longer distances.

SNMP (simple network management protocol), a standard for obtaining information and managing devices on a network.

star topology, wiring goes to a central point.

station drop, a box that contain the network jacks.

straight-through cable, a cable that just passes the signals through pin to pin (e.g. pin 1 to pin 1, pin 2 to pin 2, etc.).

sun-ether, 192.9.200, the network number that Sun used as an example in their documentation. Many sites used it as their network number.

switch, the marketing speak for multiport bridge. Usually implies high performance.

TIA-568, Telecommunication Industry Association standard for structured wiring. Describes practices needed to meet various wiring standards.

TP-DDI, FDDI over twisted pair wires.

TP-PMD, describes the part of FDDI that was changed to support twisted pairs.

traceroute, a tool for identifying the TCP/IP route from one node to another.

transceiver, connects a device to the network.

wire management, pieces of metal or plastic to guide cables around.

wiring closet, the room or space where the wiring for a floor or area goes.

Tracking Hardware Configurations in a Heterogeneous Network with syslogd

Rex Walters – IBM Microelectronics Division

ABSTRACT

Keeping track of the RAM, disk drives, and various SCSI, network, and display adaptors currently installed in dozens (or even hundreds) of workstations can be a nearly impossible task, particularly in environments where hardware is frequently "borrowed" from one workstation and moved to another, or where machines frequently come and go. This paper describes a novel method of using *syslog* (3) to allow the workstations themselves to log their current hardware configuration with a central host every time they boot. The application reuses existing tools wherever possible, and thus provides a good degree of platform independence. The programs work with nearly any UNIX system, and should be extendable to other non-UNIX (but TCP/IP enabled) systems with only modest effort.

Motivation

A question commonly asked of system administrators, but surprisingly difficult to answer, goes something like, "How many workstations do we have available that can ...?". Variations of the question such as "Which workstations have 'extra' (memory|disks|adaptors) that we can use elsewhere?" or "Where are the workstations with the most resources?" are also common.

The difficulty arises from two facts of life:

1. Any database with this information, no matter how diligently maintained, is almost certainly out of date. Such databases are often based on purchasing records, which in many cases are out of date almost as soon as the hardware is unpacked.
2. Any attempt to determine the necessary information on-the-fly is also likely to fail due to one or more machines being unavailable (they might, for example, have been temporarily located elsewhere, scavenged to the point of inoperability, sent out for repair, or simply turned off).

The conflicting goals of avoiding manual labor (maintaining a database) yet keeping current with our workstation hardware configurations led directly to the development of the software described in this paper.

Design Issues

As is so frequently the case, the software evolved from a simple cobbled together script into something altogether more complex. A number of difficult design issues arose as the software evolved, and, perhaps surprisingly, the decisions made frequently led to somewhat more elegant code.

One overriding goal was to reuse existing tools and utilities as much as possible, drawing upon the collective experience of "the net" wherever practical. The final implementation was written in *perl* (1), uses *sysinfo* (2) to extract information in a platform independent manner, and uses standard UNIX *syslog* (3) to provide, in effect, a simple client-server database capability. The use of these three tools simplified the coding task tremendously (*logconfig* comprises only a few hundred lines of *perl* code). Using applications that had already been ported to numerous platforms also made the software inherently platform independent.

After expending a great deal of effort on some rather tortuous code, we gave up trying to track each hardware configuration change, and decided to only keep the *current* configuration of each machine (and ignore previous configurations). This simplified the code significantly.[1]

Another early decision was to allow the system to track information that could only be entered interactively, not just the information that could be determined automatically. In addition to the current hardware installed, we wanted to keep track of each workstation's primary user, owner, location, primary use, and serial number – none of these could reliably be determined programmatically. Simply adding an option to have the program prompt an interactive user for the desired information proved sufficient.

Despite the temptation to add more and more information to the database, however, we studiously avoided logging anything that already had a final arbiter (we didn't, for example, log hostname aliases or IP addresses since the canonical reference for this data is DNS). We also avoided logging anything that changes too frequently or that an unprivileged user could change in the normal course of using the

[1]The final implementation has the serendipitous side effect that the local configuration file on each machine keeps a running history of the hardware changes for just that machine.

system. We decided to log the amount of completely unallocated disk space, for example, but not the amount of "free" disk space in each filesystem as returned by *df* (unallocated disk space is common under AIX since it allows filesystems to be extended dynamically).

Since our network (not surprisingly) is comprised almost entirely of IBM RS/6000 workstations, the initial script was very AIX specific. This was an incomplete solution even in our environment, however, so we strove to make the software as platform-independent as practical (but still without preventing us from logging platform-specific data). The *sysinfo*(1) program from Michael Cooper of USC provided a platform-independent mechanism for gathering hardware configuration data, but we still took pains to structure the program such that platform-specific data gathering routines could be added easily. The current implementation only supports UNIX platforms, but it's anticipated that PCs and Macintoshes may also be supported in the future (possibly through some sort of proxy mechanism).

Our network comprises only one hundred or so individual workstations, but we specified that the final implementation "must handle at least 1000 workstations without becoming unduly stressed". This was done to ensure that the software would outlive expected growth at our site as well as to ensure that it would be applicable elsewhere.

Although much of the information generated from this software may eventually end up in a relational database of some sort, the administrative complexity of setting up and maintaining a full relational database outweighed the benefits of using one. We decided to store the configuration data in an easily parsed format that would could be easily transferred to a true relational database. A simple flat file "database" maintained by syslog, with one "record" per line and "fields" separated by white-space proved quite sufficient.

We eventually came up with the somewhat novel approach of using *syslog*(3) to implement a simple client-server database. The decision to use *syslog*(3) solved a number of thorny issues such as file locking and concurrency, and provided a simple communication mechanism for logging information from a remote host. The technique is more generally applicable for any data gathered on a host-by-host basis. A previous paper by Shipley and Wang describes a system for monitoring system usage at the Jet Propulsion Laboratories that uses *syslog* in a very similar manner [4].

Implementation

The final system was written in *perl*(1) (version 5.001), and consists of three separate programs: *logconfig*, *pruneconfig*, and *prconfig*, described below.

One machine in the network is designated the *loghost*, and maintains a configuration file with entries for every machine in the network (the "master config file"). Every other workstation in the network maintains a local configuration file containing only the information about that host (the "local config file"). The master config file is essentially the logical concatenation of all the local config files in the network. Each workstation uses *syslog*(3) to log configuration information in the local config file, as well as to relay the information to *syslogd* on the loghost for inclusion in the master config file. The loghost itself maintains both a local config file containing only its own data as well as a master config file containing data for the entire network. The difficulty of distinguishing between the two types of log data seen by the loghost (locally generated and remotely received) was overcome by logging local data at a higher priority (see below).

The core of the system is *logconfig* which uses the freely available program *sysinfo*(1) and, optionally, other platform-specific utilities to determine information about a system; it also uses *syslog* to actually log the information. By default, *logconfig* only logs hardware changes since the last time it was invoked (although this behavior can be overridden with a flag to "force" the logging of all configuration information obtained during the current invocation). Every machine in the network has its appropriate boot script modified to call *logconfig* each time it is booted. *logconfig* may also be invoked by any user at any time.

```
ahab              7012/350          MEM:  192/416 MB  DISK:    992MB (20)
AIX-3.2.5 <>3250  IDs: 0943b438 000025183800  USER: Mary Buck
   7012-26-60834  LOC: C108/060              FUNCTION: client
                  "IBM 0663L12" 1.0 GB SCSI Disk Drive; "IBM97N" Color Graphics
                  Display Adapter; Diskette Drive; Standard Ethernet Adapter;
                  Standard I/O Diskette Adapter; Standard I/O Parallel Port
                  Adapter; Standard I/O Serial Port 1; Standard I/O Serial
                  Port 2; Standard SCSI I/O Controller; Token Ring Network
                  Interface; Token-Ring High-Performance Adapter;
```

Figure 1: Example output from *prconfig*

pruneconfig is run periodically from cron on the loghost to prune redundant information from the master config file (caused primarily by modifying or removing hardware from workstations, or by using the "force" flag with logconfig).

prconfig is a simple utility to present the information contained in a config file in a more human-readable format. By default, *prconfig* only reports on information in the local config file, but if the master config file is available (at our site, the master config file is available everywhere via NFS) it can optionally generate reports for the entire network. The original intent was to provide a variety of report formats for *prconfig*, but in practice the single format shown in Figure 1 proved to be sufficient for the majority of uses. For the occasional case where the report format isn't appropriate, or a more complex query of the database is desired, it is a simple matter to cobble together a "one-off" script by modifying a copy of *prconfig* (even a simple *perl* or shell one-liner is often sufficient to query the database).

The *syslog* configuration file on every client machine, `/etc/syslog.conf`, must have two lines appended:

```
local2.info        /var/CONFIG
local2.info        @loghost
```

This reflects our configuration where the *syslog* facility used by *logconfig* is "local2", data is logged at the priority "info", the local config file is `/var/CONFIG`, and the hostname (or alias) for the loghost is "loghost".

The syslog configuration file on the loghost contains:

```
local2.notice      /var/CONFIG
local2.info        /var/ALLCONFIG
```

Since only data generated by the loghost itself is logged at priority "notice", the local config file, `/var/CONFIG`, only contains information about the loghost itself. The master config file, `/var/ALLCONFIG`, contains all data logged at priority "info" *or higher*, and thus contains both the locally generated data as well as data from all the other machines on the network.

A config file, as generated by *syslog*, can be viewed as a simple flat-file database, with newlines delimiting records and whitespace delimiting fields within a record. As an example, some of the lines from our network's master config file are shown in Figure 2.

Each line is prefixed by *syslogd* with the date, the name of the host that generated the line, and the string "CONFIG:". Each line also contains fields for a "tag", an "index", and arbitrary text. After running *pruneconfig* on the config file, the triple (*hostname*, *tag*, *index*) is guaranteed to be unique for each line (record) in the file, and thus may be used as the key/index for a table in a relational database. Figure 3 shows the major portion of the code in *pruneconfig*, and illustrates the pruning algorithm (simply put, later entries supersede earlier ones).

Currently the following tags are defined:

HOSTID The identifier returned by the *hostid*(1) command on a UNIX system (the *hostid* is frequently used by license managers, but is of dubious value on an RS/6000 because there is no guarantee of uniqueness).

MODEL The model of workstation being logged. Unfortunately, *sysinfo* cannot reliably determine the model of any arbitrary system since new models are, of course, introduced all the time. This implies that platform specific routines must correctly determine and consistently label the model (this is a frequent search field for database queries like "tell me all of the RS/6000 model 380's or better with more than 128 MB of RAM").

RAM The amount of physical memory in the system.

VM The amount of virtual memory or paging/swap space in the system.

```
Dec 13 16:17:42 ryobi.raleigh.ibm.com CONFIG: FUNCTION: * client, printserver
Jan 23 10:33:36 craftsman.raleigh.ibm.com CONFIG: SERIAL: * 7012-26-77365
Apr 17 20:50:00 skil.raleigh.ibm.com CONFIG: OSLEVEL: * <>3250
Apr 21 13:42:11 decker.raleigh.ibm.com CONFIG: FREEDISK: hdisk0 800 MB
Apr 21 13:43:00 dewalt.raleigh.ibm.com CONFIG: DEVICE: sys0-sysplanar0-bbl0
"GXT150 Graphics Adapter"
Apr 23 16:40:34 speedway.raleigh.ibm.com CONFIG: HOSTID: * 1a000396
Apr 23 16:41:13 pmatic.raleigh.ibm.com CONFIG: DEVICE: inet0-fi0 FDDI Network
Interface
Apr 23 16:44:29 elu.raleigh.ibm.com CONFIG: MODEL: * 250/PowerPC
Apr 30 11:17:18 pmatic.raleigh.ibm.com CONFIG: DEVICE: inet0-fi0 *REMOVED*
```

Figure 2: Sample lines from a master config file (lines wrapped for clarity).

OSNAME The name of the operating system (e.g., "AIX" or "SunOS").

OSVERSION The revision level of the operating system (e.g., "3.2.5").

MACHINEID The "machine identification", that is, what is typically returned by `uname -m` on a UNIX system. On RS/6000s this returns a unique hardware id for that machine (most license managers for AIX use this value for nodelocked software). On Suns, this returns the machine hardware architecture (e.g., "sun4m") which is decidedly not unique.

OSLEVEL This tag is currently only meaningful on AIX workstations, and refers to the level of patches and systems software installed (e.g., on an AIX system with an OSVERSION of 3.2.5, the OSLEVEL might be ">3.2.5" if any patches to 3.2.5 had been applied, "<3.2.5" if any systems software not at the 3.2.5 enhancement level is currently installed, or even "<>3.2.5" if both cases were true). The value for this tag is determined by running the AIX *oslevel* (1) command.

TOTDISK The total amount of hard disk space available in the system (in megabytes).

FREEDISK The total amount of completely unallocated disk space on a system (probably only meaningful with operating systems like AIX that allow filesystems to grow dynamically).

DEVICE This is currently the only tag which is normally logged multiple times for a single machine (once for each device in the system). This information is determined solely from *sysinfo*. A unique index is constructed for each device described by *sysinfo* by concatenating the names of each hierarchical "parent" device. For example, in Figure 2, the graphics adaptor for "dewalt" is the first "bbl" device on system planar 0 of system 0, and is thus indexed with `sys0-sysplanar0-bbl0`.

LOCATION The physical location of the machine. This information can only be determined interactively, and thus can't be logged automatically at boot time (a user must invoke `logconfig -i` explicitly). All of the systems administration staff at our site habitually run `logconfig -i` whenever a system is moved or ownership changes (and we send out occasional messages to all users asking them to run *logconfig* interactively on the machine they are currently using).

OWNER The person or organization that owns the machine.

USER The primary user for the machine (this can be a person or a generic category of users like "admin staff").

FUNCTION The primary function or functions for this machine, typically "client", "file server", or "compute server".

```
while (<FILE>) {
    # Skip the line unless it's from logconfig
    next unless /(... \d+ \d+:\d+:\d+) (\S+) CONFIG:\s+(\w+):\s+(\S+)/;
    ($host, $tag, $index) = ($2, $3, $4);
    # Remember it (possibly replacing previous entry)
    $line{$host}{$tag}{$index} = $_;
}
foreach $host (sort keys %line) {
    foreach $tag (sort keys %{$line{$host}} ) {
        foreach $index (sort keys %{$line{$host}{$tag}} ) {
            # If last entry was a *REMOVED*, skip it
            next if ($line{$host}{$tag}{$index} =~ /\*REMOVED\*/);
            print $line{$host}{$tag}{$index};
        }
    }
}
```

Figure 3: Algorithm used by *pruneconfig*

SERIAL The serial number of the system as a whole. Sadly, this can *not* be reliably determined automatically (*logconfig* ignores any serial number returned by *sysinfo*).

An index of "*" refers to the system as a whole; any other unique string may be used to index tags with more than one iterated value (each device in a system, for example, will have a log entry with a tag of "DEVICE" but with its own unique index).

When *logconfig* is invoked interactively, it prompts the user for any information it can't determine automatically (such as its current location, owner, and serial number). It will use any previous log entries in the local config file as default values for these queries (so the user isn't obligated to re-enter information that isn't easily obtained — serial numbers, for example, are invariably placed on the most dimly lit and inaccessible surface on a machine).

Figure 3 shows the simplified flow for logconfig.

Note especially that any hardware that was previously logged in the local config file but wasn't detected during this invocation has a special log entry generated to indicate it was removed (the entry will use the same hostname, tag, and index, but will have a text value of "*REMOVED*"). Such an entry is shown in the last line of figure 2.

Without such entries, each machine would show a current configuration that was the superset of all hardware ever present in that machine. The pruneconfig script deletes "*REMOVED*" log entries as well as the entries they refer to from config files.

Since, by default, *logconfig* only logs changes to the configuration, and *pruneconfig* removes any history of changes, it's usually advantageous to only run *pruneconfig* on the master config file, and let the local config files grow without bound. In this manner, the local config files provide a history of hardware configuration changes for each machine.

Practical Experiences

logconfig has proven to be a valuable tool at our site for managing a surprisingly difficult and time-consuming task. Previous attempts to maintain the same information manually invariably led to out of date and incorrect information. Having accurate, up-to-date data at our fingertips helped us uncover a number of resources that were better allocated elsewhere.

The author has taken to carrying a printout from *prconfig* around in his day planner, replacing it with a new printout every couple of weeks.[2] It is extremely gratifying to be able to answer questions about resources immediately (and correctly!), without having to leave a meeting.

Other pleasant benefits included being able to determine a hostname given the name of a user (users are notorious for leaving hostnames out of all problem reports — previously, this was almost always the first question out of a system administrator's mouth), and having serial numbers at our fingertips when placing calls for service or upgrades (crawling behind a machine to find a serial number is a good task to have *n* people do once, rather than one person do *n* times).

Limitations and Future Plans

One design goal that wasn't completely satisfied was to handle host name changes gracefully. It is a fact of life that host names change on occasion, and since the host names are used to index our database we worried that a host name change might invalidate the data. The current system is able to detect if a serial number had previously been logged under a different host name, and complains loudly if it does, but the only mechanism for removing any data logged under an old, invalid name is to manually edit the database. Host name changes have not been a significant problem to date.

The current implementation tacitly assumes that the DNS, NIS, or /etc/hosts databases used for hostname resolution are correct and complete at all times. We assume that no two machines have the same hostname and that the loghost will log the config entry with the correct, canonical hostname (and with the *same* hostname every time). This is, of course, hysterically optimistic, but the rationale is that hostname problems usually surface pretty quickly and occur infrequently in well maintained networks, and, arguably, any errors in the database

[2]The *psbook*(1) utility available with the *psutils* package from Angus Duggan is particularly useful for printing these reports.

```
&init;
&read_local_config_file;
&log_sysinfo;                          # Log platform-independent info
&log_arch_specific('uname');           # Log platform-specific info
&log_interactive;                      # Log interactively gathered info
&log_as_removed(@in_local_config_file_but_not_logged_this_time);
```

Figure 4: Simplified flow for *logconfig*

should be relatively straightforward to untangle as the data being tracked corresponds to actual tangible hardware that can be physically inventoried. Nonetheless, this is probably the biggest weakness with this system.

Using the hostname logged by *syslog* as the primary index into the database also introduces a slight complication in designing a proxy mechanism. It's hoped that the next version of *logconfig* will allow a user to enter information for machines that can't run *logconfig* directly (PCs or workstations without network connectivity, or machines on order). Since *logconfig* will log the hostname of the proxy machine and not the hostname (or other identifier) of the desired machine, it is necessary to specially mark these proxy entries and distinguish them from normal *logconfig* entries. Probably the simplest mechanism would be to insert two additional fields for proxy lines: a special tag "PROXY" to identify it as such, and another field containing the identifier of the machine being described.

Availability

The perl scripts described in this paper are not yet available on an anonymous ftp server; the author hopes to rectify this problem soon. In the interim, please contact the author via email at "rrw@vnet.ibm.com" if you would like to receive the *perl* scripts described here.

Author Information

Rex is the senior system administrator for IBM PowerPC Embedded Processors in Research Triangle Park, NC, where he is responsible for all aspects of administration and planning for a network of approximately 100 engineering workstations. Prior to this, Rex was the systems administration manager for Gateway Conversion Technologies in Morrisville, NC. Rex spent the first eight years of his career with Mitsubishi Semiconductor America in Durham, NC, where at various times he was an ASIC test engineer, design engineer, and CAD engineer. Rex received his bachelor of science degree in electrical engineering from Virginia Tech in 1985. He may be reached electronically at rrw@vnet.ibm.com, or via surface mail at IBM Corporation, 3039 Cornwallis Road, M/S B52/060, RTP, NC 27709.

References

[Author's note: There have been several papers presented regarding the installation and tracking of *software* configurations, including [7] and [8] from last year's LISA conference.]

[1] Larry Wall and Randal L. Schwartz, "Programming Perl", O'Reilly & Associates, 1990. [Perl itself is available from a variety of sites, the official distribution location is ftp://ftp.netlabs.com/pub/outgoing]

[2] Michael A. Cooper, "Sysinfo 3.0.1", University of Southern California, source code and documentation available from ftp://usc.edu/pub/sysinfo.

[3] *syslog*(3), *syslogd*(8), *syslog.conf*(5) man pages available with nearly all versions of UNIX.

[4] Carl Shipley and Chingyow Wang, "Monitoring Activity on a Large UNIX Network with perl and Syslogd", LISA V Proceedings, 1991

[5] lscfg(1) man page available with AIX ("list configuration").

[6] *psutils* written by Angus Duggan. Excellent PostScript manipulation utilities available at any `comp.sources.misc` archive site (including gatekeeper.dec.com).

[7] John P. Rouillard and Richard B. Martin, "Config: A Mechanism for Installing and Tracking System Configurations", LISA VIII Proceedings, 1994.

[8] Paul Anderson, "Towards a High-Level Machine Configuration System", LISA VIII Proceedings, 1994.

USENIX ASSOCIATION

USENIX is the UNIX and advanced computing systems technical and professional association. Since 1975 the USENIX Association has brought together the community of engineers, scientists, system administrators, and technicians working on the cutting edge of the computing world.

The USENIX technical conferences have become the essential meeting grounds for the presentation and discussion of the most advanced information on new developments in all aspects of advanced computing systems.

The USENIX Association and its members are dedicated to:
- problem-solving with a practical bias
- fostering innovation and research that works
- communicating rapidly the results of both research and innovation
- providing a neutral forum for the exercise of critical thought and the airing of technical issues.

USENIX holds an annual multi-topic technical conference, the annual System Administration (LISA) conference, and frequent single-topic symposia addressing topics such as security, Tcl/Tk, object-oriented technologies, mobile computing, and operating systems design – as many as ten technical meetings every year. It publishes *;login:*, a bi-monthly newsletter; Computing Systems, a quarterly technical journal published with The MIT Press; and proceedings for each of its conferences and symposia. It also sponsors special technical groups and participates in various standards efforts such as IEEE, ANSI, and ISO.

SAGE, the System Administrators Guild

SAGE, a Special Technical Group within the USENIX Association, is dedicated to the recognition and advancement of system administration as a profession. To join SAGE, you must also be a member of USENIX.

SAGE activities currently include the publishing of the Short Topics in System Administration series, the first of which is Job Descriptions for System Administrators; ''SAGE News'', a regular section in *;login:*; The System Administrator Profile, an annual survey of system administrator salaries and responsibilities; co-sponsoring the LISA conference; support of working groups; encouraging the formation of local SAGE groups; and an archive site for papers from the LISA conferences and sys admin-related documentation.

Member Benefits:
- Free subscription to *;login:*, the Association's bi-monthly newsletter featuring technical articles, SAGE News, columns on tools and techniques, book reviews, summaries of sessions at USENIX conferences, snitch reports from the USENIX representative and others on various ANSI, IEEE, and ISO standards efforts, and much more.
- Free subscription to *Computing Systems*, a refereed technical quarterly published with The MIT Press.
- Access to the papers from the USENIX conference and symposia proceedings, starting with 1994, via the USENIX Online Library on the World Wide Web.
- Discounts on registration for technical sessions at all USENIX conferences and symposia.
- Discounts on the purchase of proceedings from USENIX conferences and symposia.
- Discount on the 4.4BSD Manuals, the definitive release of the Berkeley version of UNIX with a CD-ROM published by the USENIX Association and O'Reilly & Associates, Inc.
- Special subscription rates to the periodicals *UniForum Monthly*, *UniNews*, the annual *UniForum Open Systems Products Directory*, and *Linux Journal*.
- Savings on selected titles from McGraw-Hill, The MIT Press, Prentice Hall, John Wiley & Sons, O'Reilly and Associates, and UniForum.
- Discount on software from BSDI, Inc.
- Right to vote on matters affecting the Association, its bylaws, election of its directors and officers.
- Right to join Special Technical Groups such as SAGE.

Supporting Members of the USENIX Association:

ANDATACO
Frame Technology Corporation
GraphOn Corporation
Matsushita Electrical Industrial Co., Ltd.
Sun Microsystems, Inc., Sunsoft Network Products
Sybase, Inc.
Tandem Computers, Inc.
UUNET Technologies, Inc.

SAGE Supporting Members:

Enterprise Systems Management Corporation
Great Circle Associates

For more information about USENIX and SAGE, please contact:

The USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710
WWW URL: http://www.usenix.org

Phone: +1 510 528-8649
Fax: +1 510 548-5738
Email: office@usenix.org

ISBN 1-880446-73-1